طريق المحترفين لبرنامج Adobe Flex Builder 2

الفهرس

طريق المحترفين لبرنامج Adobe Flex Builder 2

المؤلف

مبرمج

مايكل نبيل اخنوخ

للأستفسار و التعليق على الكتاب ارجو ارسال رسالة على البريد التالى :

Micheal_nabil@hotmail.com

إهداء

اهدي هذا الكتاب بباقة ورد عطرة
الى زوجتى الغالية لدفعها لى لمواصلة اكمال هذة السلسلة .
و لصديقى العزيز/ شريف محمد الشاذلى
هو اول من علمنى حرفا فى برنامج الفلاش و لغة الاكشن اسكربت و له الفضل فى اهتمامى بهذا المجال.

ملحوظة :

المادة العلمية لهذاالكتاب قد تم مراجعتها و لكنة غير مسئول عن الاخطاء التى قد تحدث من سؤء التطبيق او السهو و الخطأ مع مراعاة ان الكتاب قد تم تحريرة على برنامج الورد لذلك قد تجد مسافات زائدة فى الكود لذلك فى حالة أكتشاف أى اخطأ برجاء ارسال بريد الكترونى توضح بة الخطأ و الصفحة .

Micheal_nabil@hotmail.com
٠١٠٣٥٤٦٦٠٩

جميع الاراء الموجودة فى هذا الكتاب هى اراء تعبر عن رأى الكاتب الشخصى حتى لو لم توثق بمراجع .

مقدمة

هذا الكتاب يخاطب الشخص المتمرس فى كتابة الكود و سبق لة قراءة بعض الكتب عن البرمجة و الجرافيك عموما و ارجو ان يقراء كتاب اكشن اسكربت ٢ من الصفر و كتاب الطريق من اكشن اسكربت ٢ الى اكشن اسكربت ٣ حيث كثير من المصطلحات و الشرح لمفهوم الكائنات و لغة الاكشن اسكربت عموما و لغة ال Xml كنت قد وضعت شرح لهم فى الكتابين السابقين و اعتبر ان هذا الكتاب هو الجزء الثالث من هذة السلسلة التى اتمنى نشرها كاملة فى كتاب واحد مجمع لأنى مؤمن بأنك لتفهم الجديد يجب ان تتعلم من القديم اولا.
بعض موضوعات هذا الكتاب لم تشرح فى هذة النسخة لانها تجريبية و اكتفيت بالكود فقط لكنها تم شرحها بالتفصيل فى النسخة الكاملة .

المتطلبات الخاصة بجهازك للأعداد برنامج Flex2.0

- Intel Pentium 4 processor
- Microsoft Windows XP with Service Pack 2, Windows XP Professional, Windows 2000 Server, Windows 2000 Pro, or Windows Server 2003
- 512MB of RAM (1GB recommended)
- 300MB of available hard-disk space
- Java Virtual Machine: Sun JRE 1.4.2, Sun JRE 1.5, IBM JRE 1.4.2

Mac :

- PowerPC or Mactel (1GHz or greater)
- Mac OS X 10.4.7
- 1GB of RAM recommended
- 300MB of available hard-disk space
- Java Virtual Machine: Sun JRE 1.5
- Eclipse 3.2 (plugin configuration only)

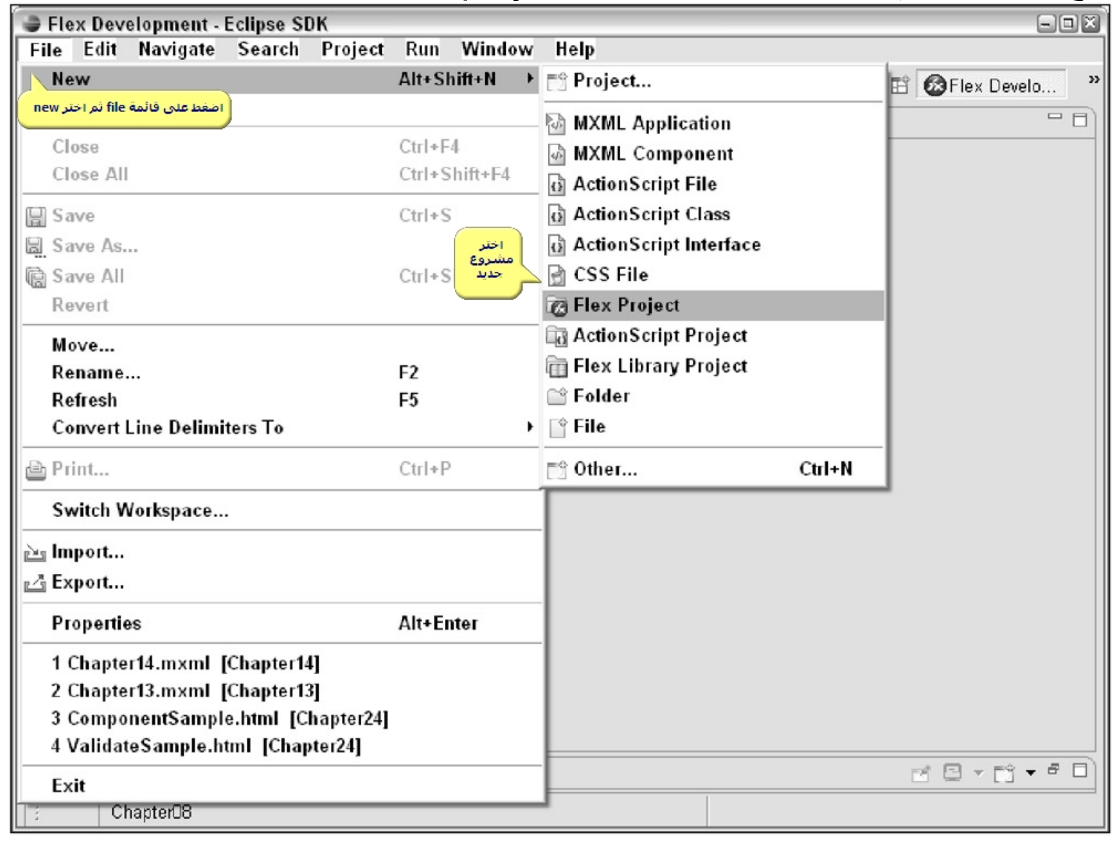
عندما نقوم بفتح البرنامج Flex 2.0 تظهر لنا الشاشة التالية



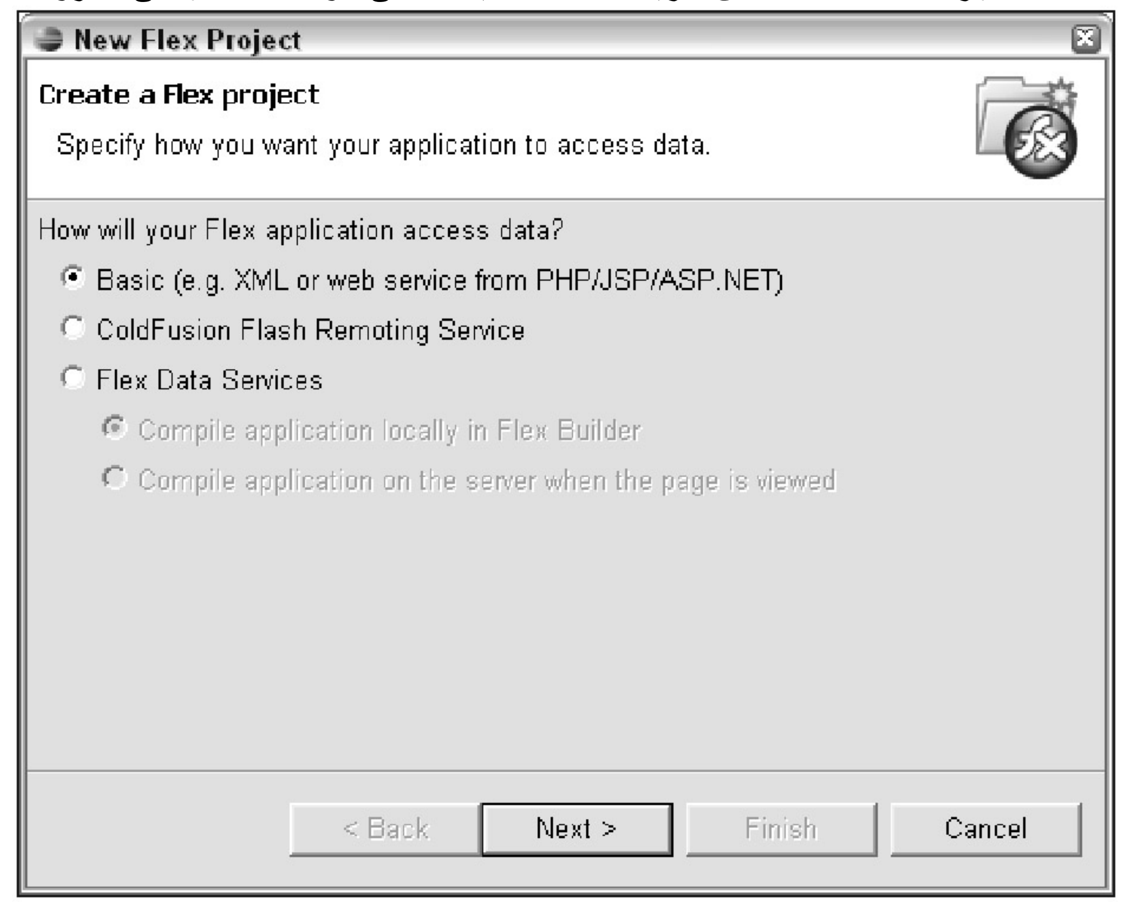
تلاحظ ان البرنامج يعرض فى البداية طريقة عملة و امثلة خاصة بة و مشاريع و مواقع على الانترنت

لعمل مشروع جديد Creating a Flex Project

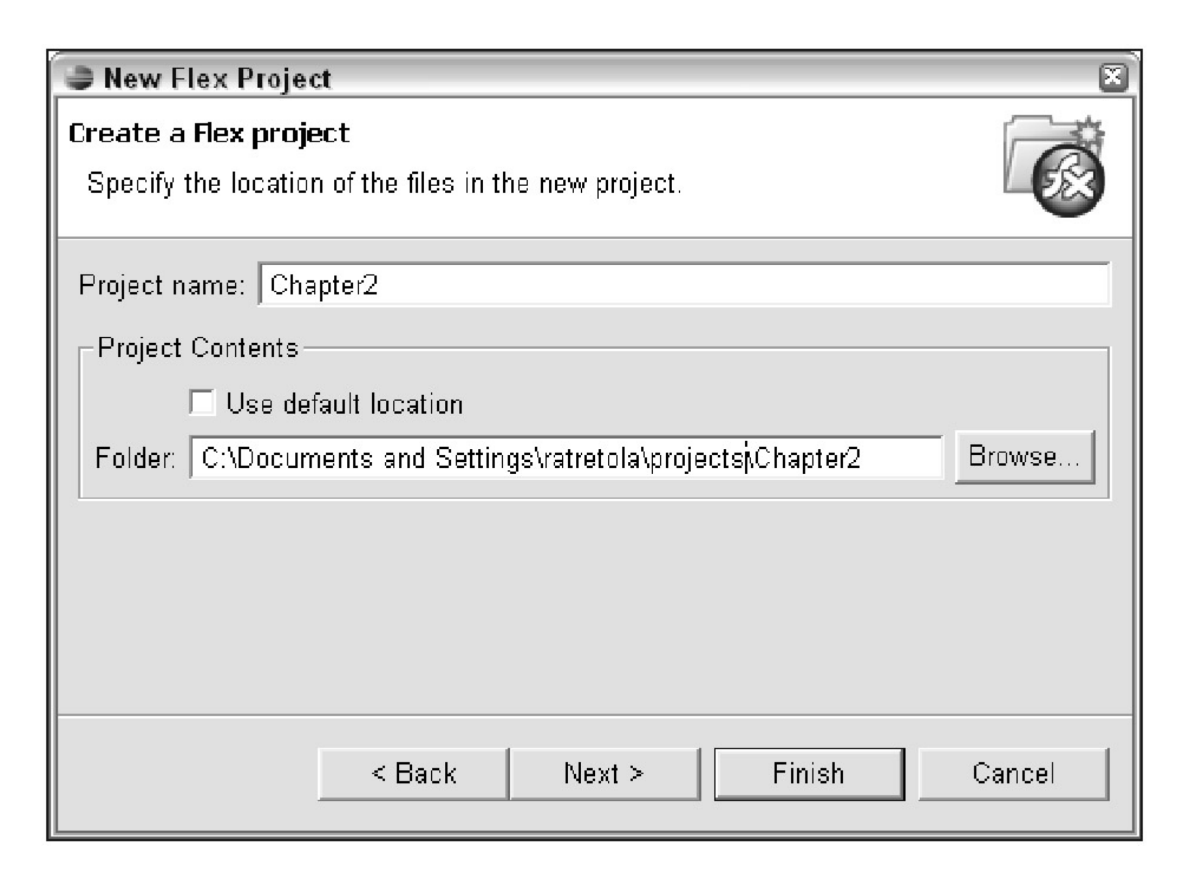
افتح قائمة File ثم أختر new و منها اختر Flex project كما فى الصورة التالية



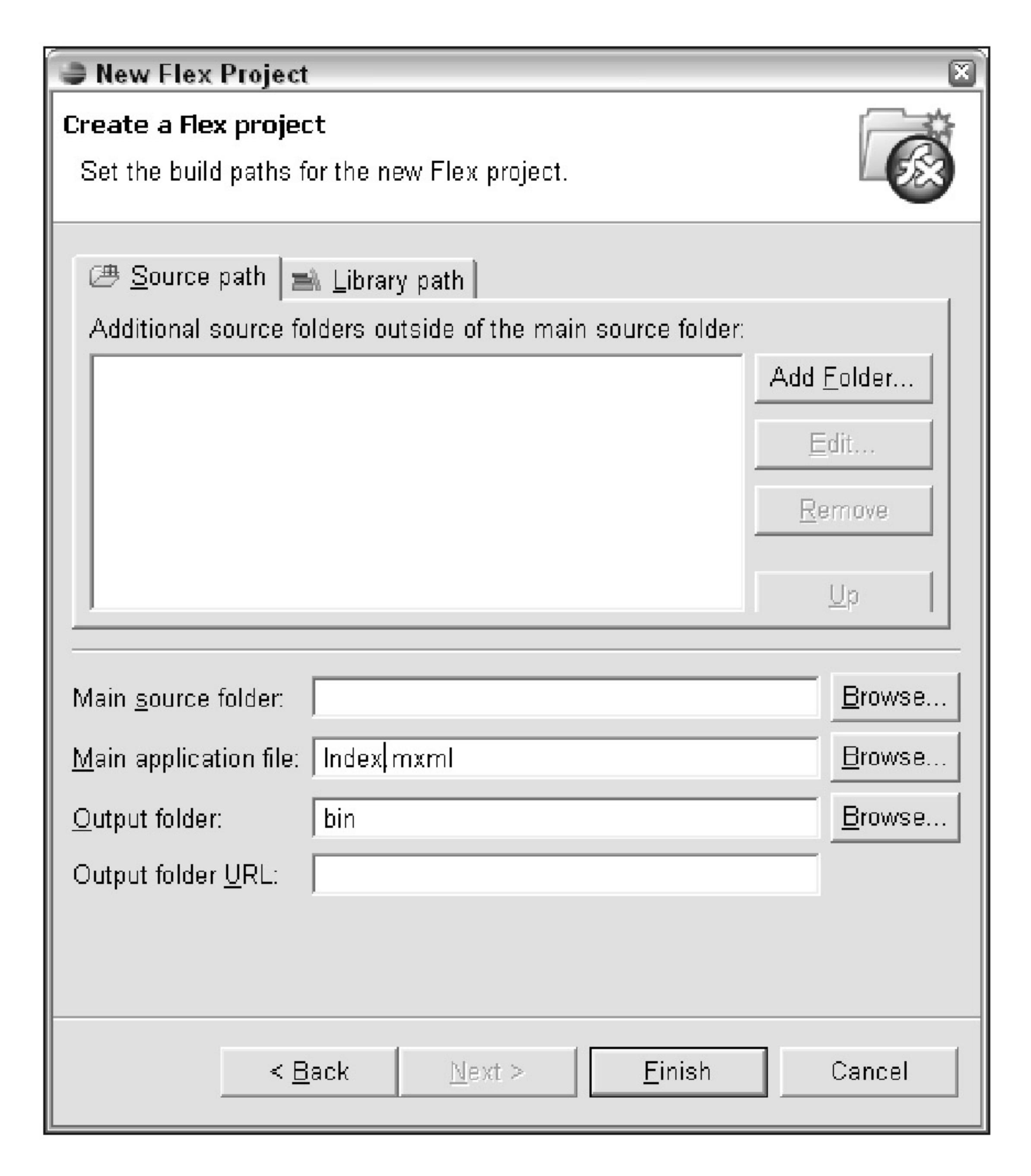
بعد ذلك تظهر لك شاشة تسالك عن طريقة اتصالك بالبيانات التى سوف تستخدمها فى مشروعك



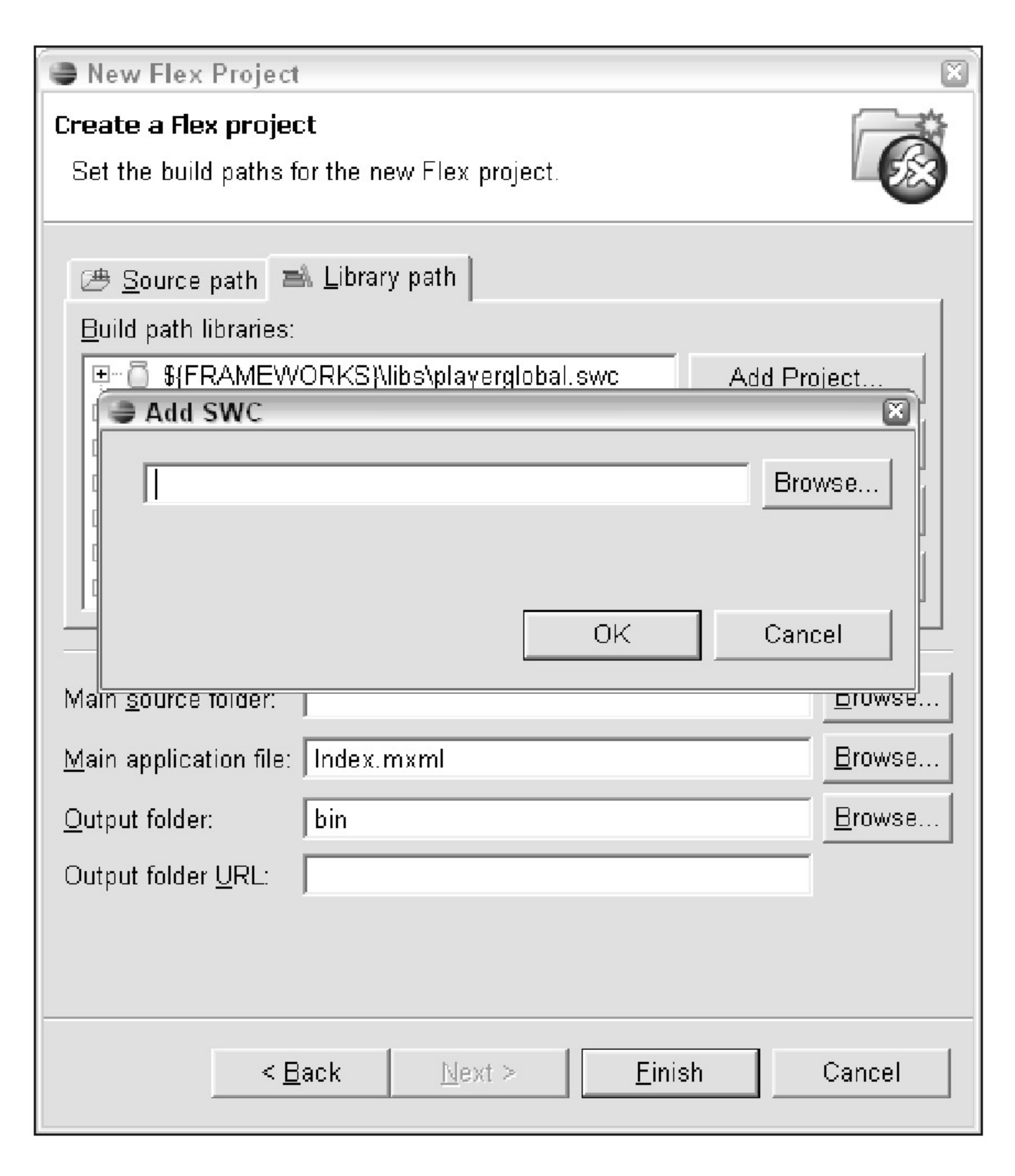
تظهر بعد ذلك شاشة تسالك عن المكان الذى تود حفظ ملفات المشروع بة و بالطبع ما هو اسم مشروعك .



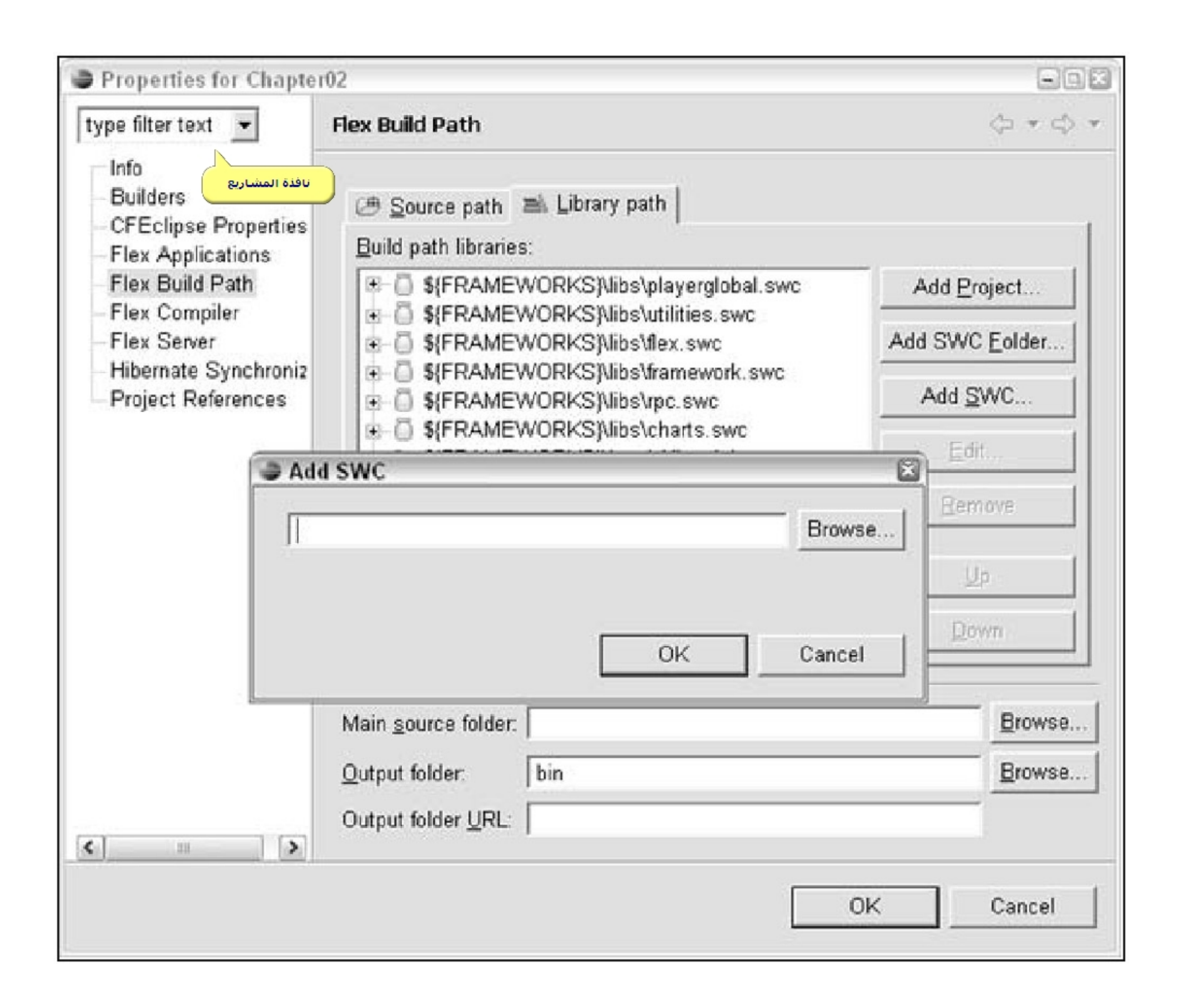
بعد ذلك تظهر شاشة تطلب منك تحديد اسم للملف الرئيسى للمشروع الخاص بك و اذا كان هناك مشاريع قديمة تود اضافتها للمشروع الحالى ام لا .



يمكنك الضغط على تبويب Library Path و تقوم بأضافة مكتبة قديمة لمشروعك SWC libraries

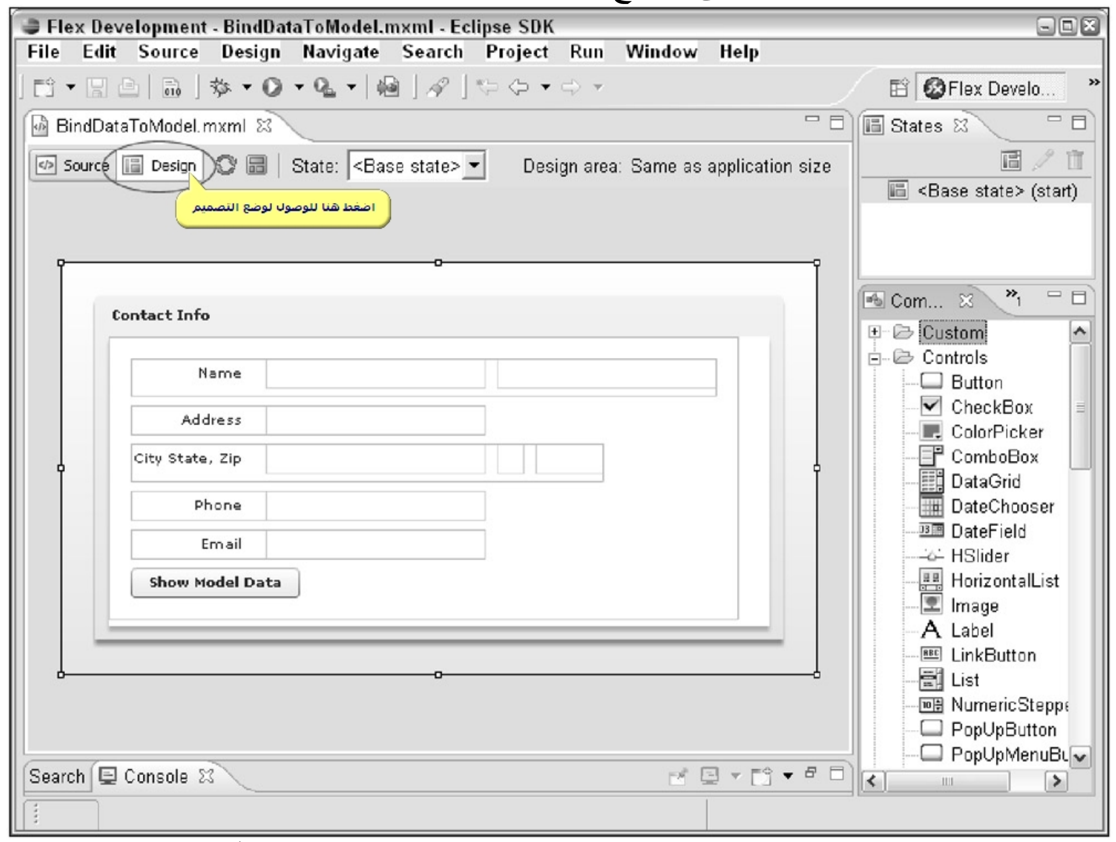


و يمكنك
اضافة مكتبة جديدة اى وقت من خلال الضغط على Add SWC Folder او Add SWC button
بعد اختيار المشروع المراد اضافة المكتبة لة من نافذة المشاريع كما بالصورة التالية

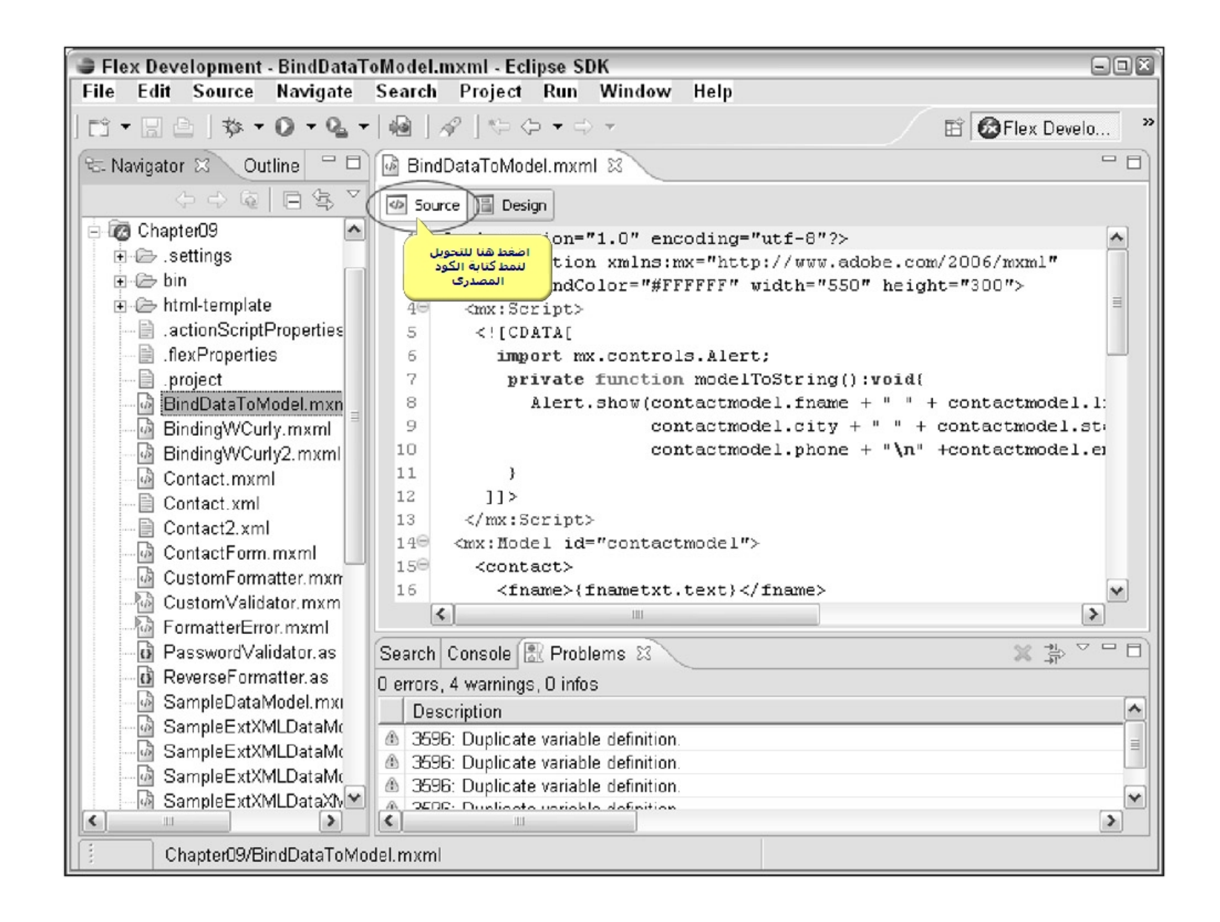

Properties for Chapter02
type filter text
نافذة المشاريع
Info
Builders
CFEclipse Properties
Flex Applications
Flex Build Path
Flex Compiler
Flex Server
Hibernate Synchroniz
Project References
Flex Build Path
Source path
Library path
Build path libraries:
${FRAMEWORKS}\libs\playerglobal.swc
${FRAMEWORKS}\libs\utilities.swc
${FRAMEWORKS}\libs\flex.swc
${FRAMEWORKS}\libs\framework.swc
${FRAMEWORKS}\libs\rpc.swc
${FRAMEWORKS}\libs\charts.swc
Add Project...
Add SWC Folder...
Add SWC...
Edit...
Remove
Up
Down
Add SWC
Browse...
OK
Cancel
Main source folder:
Browse...
Output folder:
bin
Browse...
Output folder URL:
OK
Cancel

طريقة عمل البرنامج

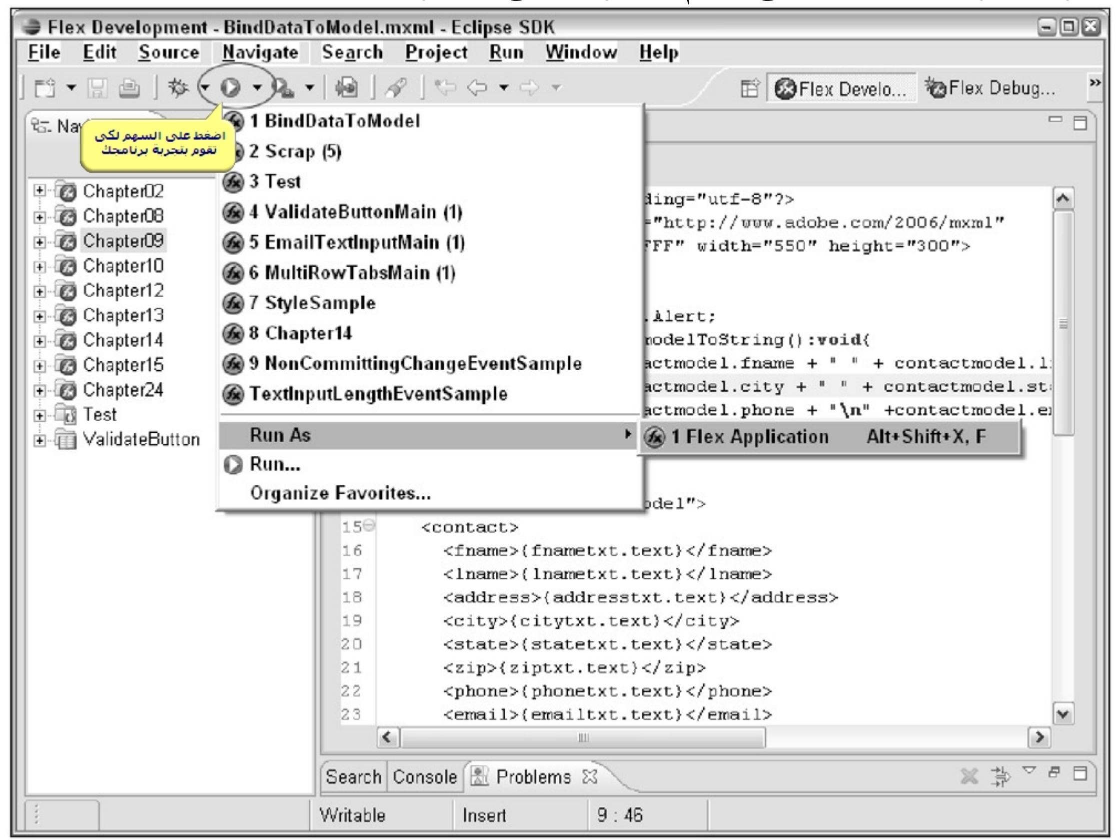
يعمل البرنامج بأسلوبين
١- نمط التصميم : حيث فى هذا النمط يمكنك تصميم شكل البرنامج الخاص بك أى الواجهة التى يراها مستخدمى البرنامج الذى تقوم بصنعة و تستطيع من خلالها التحكم فى الالوان و اشكال الأزرار و كافة الادوات على مسرح العمل .



٢- نمط كتابة الكود : هى شاشة كتابة الكود الذى ينفذ بالضغط على احد الأزرار فى برنامجك أو أي حدث فى برنامجك .



لتجربة مشروعك اضغط على السهم المشار الية فى الصورة التالية



The elements of a Flex application

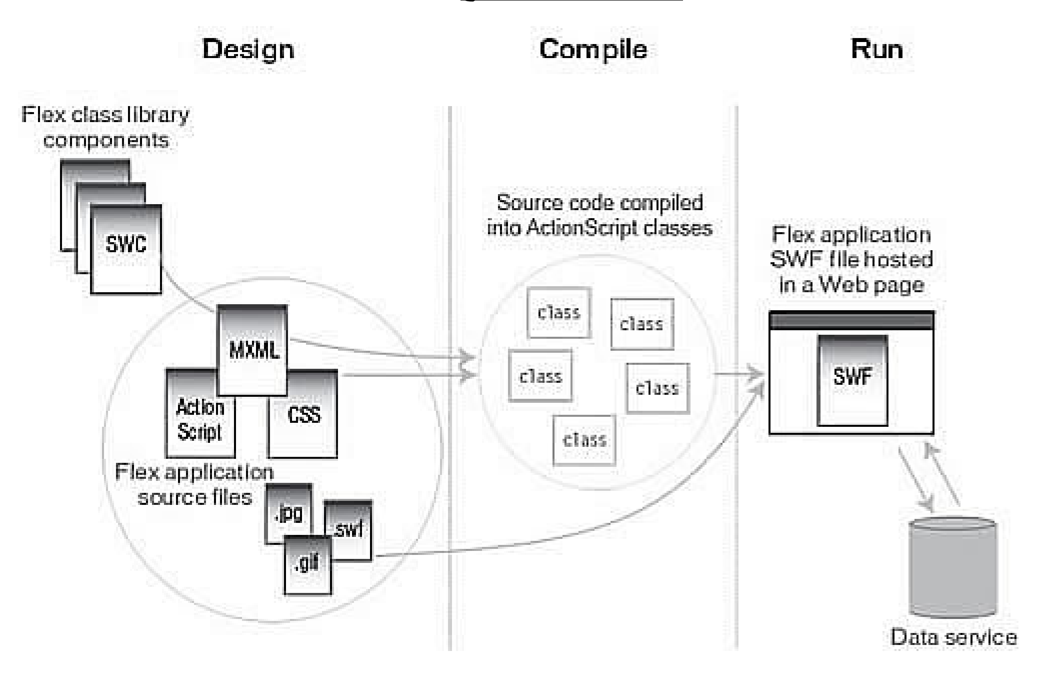
العناصر التي يتكون منها المشروع داخل برنامج Flex

اى مشروع يجب ان يحتوى على العناصر التالية :

Flex framework	بيئة اطار العمل الخاصة ببرنامج The Adobe® Flex 2 و التى تحتوى على كل الكمبونات او الادوات اللازمة لبناء موقع على الانترنت او برنامج بصفة عامة و المقصود ان البرنامج يوفر لك واجهة مليئة بالادوات المستخدمة فى جمع البيانات من المستخدم مثل مربعات الحوار و الازرار و اداوات عرض البيانات و وسائل التأكد من صحة الادخال و كثير من المؤثرات و كل هذا يرتبط بالمكتبة component library (SWC) file .
MXML	كل مشروع يحتوى على الاقل ملف واحد MXML file معرف فى بداية المشروع لاحظ ان ملف MXML يعتبر لغة قريبة فى تركيبها من لغة ال html و لغة ال xml بالطبع . و هذة اللغة صممت خصيصا لهذا البرنامج لوصف بناء المشاريع من خلال الوسوم tags.
ActionScript 3.0	لكى تقوم بعمل تطبيقات تتفاعل مع المستخدم يجب ان تستخدم لغة ActionScript 3.0 و هى شبيهة فى تكوينها من لغة الجافا اسكربت و يمكنك اضافة اكواد الاكشن اسكربت ٣,٠ داخل البرنامج مباشرة من خلال ملف ال MXML بين tags او فى ملف منفصل و عمل استيراد لة داخل المشروع .
CSS	جداول الانماط المتعاقبة و المقصود بها عمل شكل خاص بالازرار او الادوات المستخدمة فى النماذج
Graphic assets	مثل معظم البرامج يمكنك استخدام صور و ايقونات و مؤثرات
Data	بعض الادوات تستخدم لعرض البيانات مثل (a combo box or data grid for example) و ذلك من خلال المصفوفات او مصدر خارجى للبيانات مثل ملفات النصوص txt او xml

How Flex applications are compiled and deployed

فكرة عمل البرنامج



اولا : مرحلة التصميم من خلال خلايا جاهزة فى البرنامج و يمكنك عمل خلايا و مكونات خاصة بك باستخدام كود الاكشن اسكربت .

ثانيا : مرحلة الترجمة او تحويل الكود المكتوب بلغة الاكشن اسكربت الى خلايا .

ثالثا : مرحلة تنفيذ الكود و اعداد ملف ال SWF و ربطة مع البيانات الخارجية .

يمكنك عمل النماذج الخاصة بالبرنامج الخاص بك من خلال كود mxml و هذا مثال لعمل لوحة بها نص و اضا زر للأغلاقها:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute">
<mx:Panel>
<mx:TextArea text="Say hello to Flex!" />
```

```
<mx:Button label="Close" />
</mx:Panel>
</mx:Application>
```

اذا كنت معتاد على لغة ال xml فسوف تلاحظ التنسق الخاص بهذةاللغة و يمكنك الرجوع الى الملحق الخاص بلغة الxml فى كتابى السابق الطريق من اكشن اسكربت ٢ الى اكشن اسكربت ٣ لكى تفهم الفكر الاساسى لهذة اللغة ..

المهم ان كل مشروع سوف تقوم بعملة داخل برنامج Flex2

سوف يحتوى فى المقدمة على كود

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute">
```

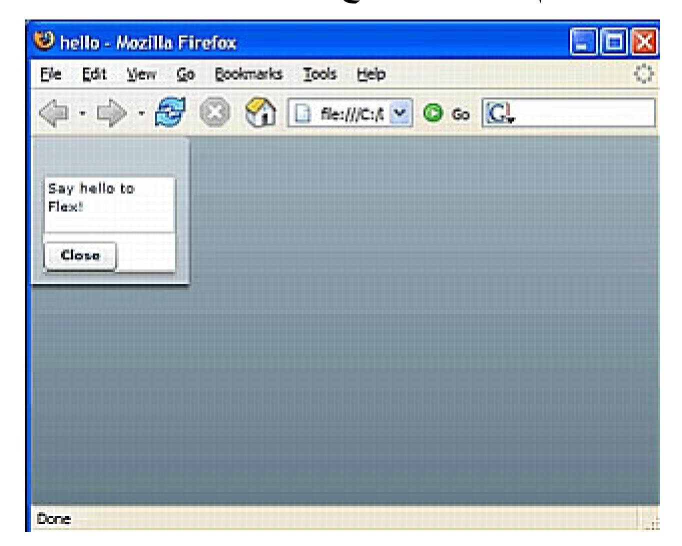
و يجب ان ينتهى المشروع بالكود

```
</mx:Application>
```

اما الاكواد الخاصة برسم مربعات النص و الزر و غيرها تكتب فى المنتصف بين الكودين السابقين .

لاحظ ان : كل تاج فى ملف mxml يبدأ بحرفى mx و الذى يعتبر namespace صمم خصيصا لبرنامج flex.

عندما تقوم بتشغيل البرنامج سوف تشاهد التالى



لاحظ ان مشروعك فى النهاية ليس سوى ملف فلاش بامتداد swf و تشاهدة من خلال مستعرض الويب الخاص بك و هنا نحن نستخدم برنامج Mozilla firefox .

MXML

ستلاحظ ان معظم اكواد MXML تتعلق بكلاسات AS 3.0 او خصائص لها حيث ان مترجم برنامج Flex يرسل اكواد MXML على هيئة swf بايت يتكون منها فيلم الفلاش فى النهاية ليصبح swf .

الخلاصة :
لن يمكنك كتابة أكواد اكشن اسكربت ٣ مباشرة داخل نافذة تحرير الكود بل يجب أن تكتبها بين تاجين خاصين كما يلى

```
<mx:Script>
   <![CDATA[
      ...
   ]]>
</mx:Script>
```

طريقة استخدام اكشن اسكربت داخل البرنامجUsing ActionScript

• Inline within MXML tags

فى نفس السطر او داخل تاج MXML مثال اظهار رسالة عند ضغط المستخدم على زر

```
<mx:Button id="alertButton" label="Show Alert"
click="mx.controls.Alert.show('Example')" />
```

فى المثال استخدمنا الدالة Show المنشقة من الخلية Alert فى لغة الاكشن اسكربت لعرض الرسالة داخل الاقواس .

• Nested within MXML tags

يمكنك كتابة الكود متداخل مع اكواد Mxml و تستخدم تاج خاص للتعبير عن ما بداخل اهذا التاج هو كود اكشن اسكربت

CDATA block

مثال

```
<mx:Button>
<mx:click>
<![CDATA[
mx.controls.Alert.show("Example");
]]>
</mx:click>
</mx:Button>
```

• In MXML scripts

تكتب الكود داخل تاج <mx:Script> ثم تاج <![CDATA[و نقفل التاجين ب]] و </mx:Script>
مثال

```
<mx:Script>
<![CDATA[
import mx.controls.Alert;
private function example( ):void {
Alert.show("Example");
```

```
}
]]>
</mx:Script>
```

• Within ActionScript classes

يمكنك كتابة اكواد الاكشن اسكربت فى خلايا و ارفاقها مع البرنامج و استدعائها من خلال الكود التالى

```
<mx:Script source="code.as" />
```

MXML and ActionScript Correlations

الارتباط بين اكشن اسكربت و MXML

عندما تستخدم MXML لعمل زر مثلا فان هذا الحدث يتساوى مع استدعاء لغة الاكشن اسكربت للخلاية السئولة عن انشاء زر و لنتفهم الامر اكثر سوف اوضح بالكود

```
<mx:Button id="button" />
```

يتساوى الكود السابق مع كود الاكشن اسكربت التالى

```
var button:Button = new Button( );
```

ايضا مثال اخر عن اسناد قيمة لخاصية النص فى زر

```
<mx:Button id="button" label="Click" />
```

و ذلك يعادلة الكود التالى فى الاكشن اسكربت

```
var button:Button = new Button( );
button.label = "Click";
```

نفهم من السابق ان اكواد MXML فى جوهرها ليست الا استدعاء لخلايا من لغة الاكشن اسكربت و لكن فى شكل مختلف عن طريقة كتابة الكود التى تعودنا عليها فى لغة الاكشن اسكربت .

و لكن لاحظ انك قبل ان تستخدم كود الاكشن اسكربت مع اى اداة صممتها بلغة MXML يجب ان تعطيها ID اى اسم خاص بها لتستطيع التعامل معها مثلا

```
<mx:TextInput id="myTextInput" text="Refer to me via the id
property" />
```

و لكن خاصية ال ID خاصية اختيارية و لكن اجبارية فى حالة اذا كنت تريد ان تستخدم هذة الاداة مع الاكشن اسكربت حيث ان مترجم MXML يقوم بعمل متغير يدعى myTextInput يحتوى بداخلة ارتباط مرجعى للكائن TextInput مثال على ربط الاداة المصممة بلغة MXML بكود الاكشن اسكربت

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    verticalAlign="middle"
    horizontalAlign="center"
    xmlns="*">

 <mx:Script>
        <![CDATA[

            public function getText():String
```

```
            {
                return myTextInput.text;
            }

        ]]>
    </mx:Script>

 <mx:TextInput id="myTextInput" text="Refer to me via the id
property" />

</mx:Application>
```

الدالة getText() ترجع بقيمة نصية اسندت للمتغير myTextInput

ملحوظة : يجب ان يكون جميع قيم ID الموجودة داخل ملف واحد متفردة اى لا تتكرر .

- و يمكن اسخدام كلمة This للأشارة للأداة اذا كانت ليس لها ID فى الملف

مثال :

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    verticalAlign="middle"
    horizontalAlign="center">

    <mx:Script>
        <![CDATA[

            import mx.controls.Alert;
            import mx.controls.TextInput;

            public function getText():String
            {
                return this["myTextInput"].text;
            }

        ]]>
    </mx:Script>

 <mx:TextInput id="myTextInput" text="Hello World!" />
 <mx:Button label="Get Text" click="Alert.show(getText())" />

</mx:Application>
```

Understanding ActionScript Syntax

فهم اسلوب لغة الاكشن اسكربت

مفهوم الحزم البرمجيةUnderstanding Packages

ان معظم الخلايا وضعت منظمة فى تركيب يدعى حزم برمجية و لكى تفهم لغة الاكشن اسكربت يجب ان تفهم مفهوم الحزم .
ان الحزم البرمجية هى تجميع لمجموعات من الخلايا المتقاربة فى الغرض لكى يمكنك استخدامها داخل برامجك فى نطاق Scope مجال معين .
و تتسأل الان ما هو Scope ؟
انة مفهوم يوضح مدى رؤية الكائنات و المتغيرات بالنسبة لمراحل حياة البرنامج الرئيسى و الخلايا التى يتكون منها و لنفترض مثلا انك لديك متغير فانك من خلال معرفتك مدى او مجالة يمكنك ان تعرف اين هو بالضبط داخل الكود و كيف تستدعية و تسند لة قيمة اخرى .
و لكن ما فائدة ال Scope بالنسبة للحزم البرمجية ؟
الحزم البرمجية تسمح لك بعمل مجموعة من الخلايا بنفس الاسم لكن بشرط استخدامهم فى مجالات مختلفة اى Different Scope.
مثال Button Class هى جزء من الحزمة البرمجية mx.Controls اذن المسار الكامل للزر هو

```
mx.controls.Button
```

و علية فاذا كنت تريد عمل خلية زر اخرى فى حزمة برمجية اخرى دون ان تخاف ان يحدث تعارض بين الخلية الجديدة و الخلية القديمة mx.controls.Button
مثال على تعريف زر من الخلية Button فهناك اكثر من طريقة منها :

```
var button:mx.controls.Button;
```

او باستخدام Constructor

```
button = new mx.controls.Button( );
```

و يمكنك ايضا تعريف متغير من النوع زر من خلال استيراد الخلية زر

```
import mx.controls.Button;
```

و لكن فى هذة الحالة اذا قمت بتعريف زر اخر من خلية اخرى يجب ان تتعامل مع كلا منهما بالاسم الكامل اى بهذة الطريقة

```
var button:mx.controls.Button;
```

الاعلان عن الخلايا Declaring Classes

فى البداية يجب ان تقوم بعمل ملف يحتوى على كود الخلية و يكون امتداد هذا الملف as و يشترط ان يكون اسم الملف هو نفس اسم الخلية مثلا اذا كنت تريد عمل خلية اسمها Example فيجب ان يكون اسم الملف الخاص بها هو Example.as
مثال :

```
package com.example {
import flash.net.URLLoader;
import flash.net.URLRequest;
public class Example {
// الاكواد الخاصة بالخلية تكتب هنا
}
}
```

الاعلان عن الحزم البرمجية Package declarations

يرتبط الاعلان عن حزمة برمجية بالمسار الموجود بة ملف الخلية على القرص الصلب فى جهازك مثلا اذا كان لديك ملف خلية اسمة Example. as داخل مجلد اسمة Com فان المسار لهذا الملف يكون كالتالى Com.Example و يفصل بين اسم المجلد و اسم الخلية نقطة .

مثال : للأعلان عن حزمة برمجية

```
package com.example {
// Import نستخدم هنا جملة
// Class declaration نكتب هنا تعريف الخلية
}
```

تستخدم عادة عند تعريف الحزمة البرمجية و لكن لا تستخدم عند Import statements تعريف الخلية

مثال على استدعاء خلايا URLLoader و URLRequest من الحزمة flash.net

```
package com.example {
import flash.net.URLLoader;
import flash.net.URLRequest;
// Class declaration goes here.
}
```

المتغيرات و الخصائص Variables and Properties

المتغيرات Variables : هى اسماء لعناصر يمكنك اسناد قيم او بيانات لها و التعامل معها من خلال الاسم الذى اطلقتة عليها .

و لكى تتعامل مع متغير يجب ان تعرف كيفية الاعلان عن متغير و ذلك باستخدام كلمة Var لحجز مكان فى ذاكرة الجهاز لهذا المتغير مثال

```
var variableName;
```

و يفضل تحديد نوع المتغير فى جملة الاعلان عنة و الشكل العام لهذا الكود هو

var *variableName*:*DataType*;

Var اسم المتغير : نوع المتغير ;

و نوع البيانات المقصوبة بة هو اى نوع من انواع البيانات يحتوى هذا المتغير

انواع البيانات

Data type نوع البيانات	الشرح	Description
String	حرف او اكثر و يتضمن نظام الحروف الدولى الموحد	One or more characters, including all Unicode characters

Number	اى قيمة عددية متضمنا القيم الكسرية	Any numeric value, including floating-point numbers
int	الاعداد الصحيحة الموجبة و السالبة	Positive and negative integers and 0
uint	الاعداد الصحيحة الموجبة و الصفر	Positive integers and 0
Boolean	صح او خطأ قيم منطقية	True or false
Date	تاريخ ووقت	The date and time
Array	مجموعة من البيانات المنظمة و لها فهرس مرتب	An index-ordered collection of data

مثال على تعريف متغير من النوع نص String

```
var userName:String;
```

بعد ما تعرف متغير يجب ان تسند لة قيمة من خلال علامة = مثال :

```
userName = "michael";
```

و عندما تريد استرجاع قيمة المتغير و اسنادها لمربع نص تكتب الكود التالى :

```
textInput.text = userName;
```

ملحوظة :المتغيرات تعرف داخل الدوال الخاصة بالخلايا سوف نتكلم لاحقا عن الدوال . اما المتغيرات التى تعرف خارج الدوال تدعى خصائص و هذا يرجع لمجال رؤيتها داخل الخلية و فى معظم الاحوال المتغيرات و الخصائص هما نفس الشئ مع اختلاف المجال او المدى للتعامل معهما اى المسائلة مسائلة Scope

و نحصر موضوع المجال Scope بين المتغير و الخاصية فى النقط التالية :

- كل المتغيرات التى يتم الاعلان عنها داخل دالة يعتبر مجال رؤيتها داخل هذة الدالة و هذا معناة انة لا يمكنك الاشارة الى هذة المتغيرات من خارج هذة الدالة .

- على الطرف الاخر نجد الخصائص لها مجال رؤية اكبر على الاقل هى مرئية داخل الخلية كلها و يمكن الاشارة اليها من خارج الخلية بواسطة اسناد محددات لمجال الخلية هى

1- public عام

تعنى ان الخصائص التى تم تعريفها فى هذة الخلية يمكن الرجوع اليها و التعامل معها من خارج الخلية

2- private خاصة

لا يمكن التعامل مع الخصائص الا من داخل الخلية .

3- protected محمية

هذة الخصائص محمية داخل الخلية و لا تستخدم الا من داخل الخلية او من خلية ورثت هذة الخصائص من الخلية الام .

4- Internal داخلية

الخصائص يمكن الوصول اليها من داخل الحزمة البرمجية فقط .
من الناحية العملية اعتقد انة من الافضل ان تقوم بالاعلان عن الخصائص اما محمية protectedاو خاصة Private لان الخلايا يجب ان تتعامل دائما مع قيم الخصائص الخاصة بها و علية حاول قدر الامكان عدم استخدام public او Internal

- يمكنك الاعلان عن الخصائص داخل الخلايا بنفس اسلوب الاعلان عن المتغيرات و لكن يمكنك تمييز الفرق بين الخصائص و المتغيرات من خلال عملاشارة خاصة بك فى الكود عند تسمية الخصائص مثلا استخدم علامة الشرطة _ مثلا عند تسمية خاصية فأنك تكتب الكود التالى :

```
package com.example {
import flash.net.URLLoader;
import flash.net.URLRequest;
public class Example {
private var _loader:URLLoader;
}
}
```

- لاحظ قمت بتسمية الخاصية _loader من النوع URLLoader
- بالاضافة الى المحددات السابقة عام و خاص و محمى وداخلى يمكنك اضافة المحدد ساكن Static الذى يسمح لك بالوصول المباشر للخاصية من داخل الخلية بدلا من عمل عنصر من الخلية او بمعنى اخر اشتقاق كائن من الخلية مثال :

```
package com.example {
import flash.net.URLLoader;
import flash.net.URLRequest;
public class Example {
private var _loader:URLLoader;
static private var _instance:Example;
}
}
```

- هناك ايضا مصطلح خاص بالخصائص هو Constant اى الثابت و هو مثل المتغيرلكن لا يمكنك تغيير قيمتة لانها ثابتة و من الجدير بالذكر انك تعاملت مع الثوابت الخاصة ببرنامج الفلاش و منها

Event.COMPLETE, MouseEvent.CLICK, TimerEvent.TIMER, and Math.PI.

- و غالبا تعرف مجالات الثوابت على انها ساكنة او عامة مثال

```
package com.example {
import flash.net.URLLoader;
import flash.net.URLRequest;
public class Example {
private var _loader:URLLoader;
static private var _instance:Example;
static public const TEST:String = "test constant";
}
}
```

}

Methodsالدوال

الدوال : هى تجميع عدد من الاوامر البرمجية تتكرر كثيرا فى برنامجك و اسناد اسم لها لكى يريحك من عناء كتابة نفس الكود اكثر من مرة فيكفى استدعاء الاسم لتنفيذ هذة الاوامر و بعض هذة الدوال يكون لها معاملات اى قيم ترسل للدالة و ترجع بالناتج .
مثال

```
function test( ):void {
}
```

الدالة السابقة لا تأخذ معاملات و لا ترجع ببيانات لذلك ليس لها معنى و لكن اذا قمت بتغيير الكود السابق الى الكود التالى

```
function test( ):void {
var message:String = "function message";
trace(message);
}
```

فى المثال السابق الدالة test قمنا بتعريف متغير داخلها من النوع String و اسمة messageو يحمل القيمة function message و تقوم الدالة بارجاع قيمة المتغير message من خلال الدالة trace و هى المسئولة عن عرض شاشة للمخرجات مشابهة لنافذة الدوس فى لغات البرمجة الاخرى .
و عند استدعاء الدالة يكفى ان تكتب اسمها

```
test( );
```

مثال على دالة تأخذ معاملات او بمعنى اخر نرسل لها قيم لتقوم بعمل عمليات ليها و تعود بالناتج

```
function test(a:String, b:String):void {
trace("Your message is " + a + " and " + b);
}
```

و عندما نستدعى هذة الدالة نستدعيها بالشكل التالى

```
test("one", "two");
```

لاحظ ان الدوال تستخدم نفس المحددات للمجال

public, private, protected, internal, and static

مثال على تعريف دالتين احدهما public و الاخرى Static public

```
package com.example {
import flash.net.URLLoader;
import flash.net.URLRequest;
public class Example {
private var _loader:URLLoader;
static private var _instance:Example;
static public const TEST:String = "test constant";
public function traceMessage(message:String):void {
trace("Your message is " + message):
}
static public function getInstance( ):Example {
if(_instance == null) {
```

```
_instance = new Example( );
}
return _instance;
}
}
}
```

و الدوال على العكس من الخصائص من المستحب ان تكون عامة public.

- الخلايا ايضا لها نوع خاص من الدوال يدعى Constructor و لة الخصائص التالية :

١- اسم الدالة يجب ان يكون هو نفس اسم الخلية

٢- الدالة يجب تعريفها كدالة عامة public

٣- الدالة لا يجب ان يعلن بها عن بيانات راجعة او قيم مرتجعة من هذة الدالة .

المثال التالى يوضح كيفية اسناد الConstructor لقيمة جديدة للخاصية _loader :

```
package com.example {
import flash.net.URLLoader;
import flash.net.URLRequest;
public class Example {
private var _loader:URLLoader;
static private var _instance:Example;
static public const TEST:String = "test constant";
public function Example( ) {
_loader = new URLLoader( );
}
public function traceMessage(message:String):void {
trace("Your message is " + message);
}
static public function getInstance( ):Example {
if(_instance == null) {
_instance = new Example( );
}
return _instance;
}
}
}
```

و قبل ان انهى حديثى عن الدوال يجب ذكر نوعين من الدوال هما *getter* و *setter* و *يتم الاعلان عنها كأنها دوال و لكن يسهل التعامل معهم كانهم خصائص عامة*

و الاعلان عن هذة الدوال مثل الدوال الاخرى و لكن هناك اختلافات هى :

١- Getter method تستخدم get Keyword

٢- Getter method تستخدم set keyword

٣- Getter method لا تأخذ معاملات و يجب ان ترجع الدالة بقيمة .

٤- Setter method يجب ان يكون لها معامل واحد و يجب ان يعرف void return type

و المثال التالى يوضح استخدام

```
package com.example {
import flash.net.URLLoader;
import flash.net.URLRequest;
public class Example {
private var _loader:URLLoader;
static private var _instance:Example;
private var _sampleProperty:String;
public function get sampleProperty( ):String {
return _sampleProperty;
}
public function set sampleProperty(value:String):void {
_sampleProperty = value;
}
static public const TEST:String = "test constant";
public function Example( ) {
_loader = new URLLoader( );
}
public function traceMessage(message:String):void {
trace("Your message is " + message);
}
static public function getInstance( ):Example {
if(_instance == null) {
_instance = new Example( );
}
return _instance;
}
}
}
```

و عند استدعاء هذة الدالة

```
var example:Example = new Example( );
example.sampleProperty = "A";
// A عند استدعاء Setter نعطيها معامل مثلا
trace(example.sampleProperty); // Call the getter
```

الجمل البرمجيةStatements

هى الجمل و المعادلات التى يتم كتابتها فى برامجنا مثلا عندما تقوم بتعريف متغيرين و تريد جمعهم مثلا فان الجملة البرمجية فى هذة الحالة هى:

```
total = unitValue * quantity;
```

و ممكن تكون جملة بها استدعاء لدالة

```
trace("This is a simple statement.");
```

جملة التكرار Looping statements

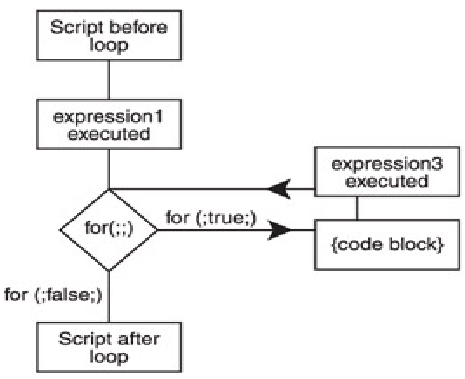
قدرة الكمبيوتر على تكرار أي جزء من الكود ـ خاصّة مع سرعته الفائقة ـ هي ما تجعله مريحا جدا للبشر، ليحمل عنهم عناء الرتابة والبطء والملل

جملة التكرار "من إلي" ...For:

الدوران او التكرار او عمل looping من الاوامر الاساسية في جميع لغات البرمجة فمثلا اذا كان البرنامج سيدخل اسماء الف موظف هل تعتقد انك ستكتب الف امر لادخال هذه الاسماء بالطبع ستكون حماقه ولكن لو وضعنا امر واحد فقط لادخال اسم الموظف وطلبنا من البرنامج الدوران الف مرة حول هذا الامر بالطبع سيكون شئ جميل ان يدخل الف بيان بمجهود بسيط نتيجة تسهيل اعطته لغة البرمجة

مثال

```
for(i = 0;i < 1000; i++){
   trace(i);
}
```



انشائنا عداد يعد من صفر الى ١٠ ورمزنا لها بالرمز i ثم داخل العداد طلبنا منه ادخال الموظف رقم i وهو عداد متغير حتى يكتمل العداد بوصوله للالف ويكون قد تم تنفيذ الامر معه الف مرة بأدنى مجهود وكلما تغير العد من ١ الى ٢ الى ١٠٠٠ تغير معه رقم الموظف بنفس الطريقة.

الجدول التالى يوضح صيغ التكرار المختلفة الخاصة بـFor وتفسيرها.............

الصـيـغـة	التـفـسـيـر
for	امر اللغة لعملية التكرار
(	قوس مفتوح يوضح بداخله بارامترات الامر
i=0;	المتغير = رقم بداية الحلقات

شرط نهاية الحلقات	i<1000;
المتغير يزيد بمقدار واحد مع بداية كل حلقة – لاتضع بعده فاصلة منقوطة	i++
قوس نهاية بارامترات الامر – لاتضع بعده فاصلة منقوطة	)
قوس بداية بلوك الاوامر المطلوب تكرارها	{
بداخل اقواس البلوك توضع الاوامر المطلوب تكرارها	
قوس نهاية بلوك الاوامر المطلوب تكرارها	}

بداخل بلوك الاوامر تم تنفيذ الامر

وهو امر يقوم بطبع قيمة x التي تتغير في كل مرة ابتداء من صفر حسب ماذكرت ان

i=0 وتزيد في كل مرة بمقدار ١ حسبما ذكرت ان i++ وذلك حتى يصل الى ٩٩ حسبما

ذكرت ان i<100

وبالتالي ستكون مخرجات البرنامج كما يلي

0

1

2

3

4

.

.

.

99

مثال أخر :

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    verticalAlign="middle"
    horizontalAlign="center"
    xmlns="*">

    <mx:Script>
        <![CDATA[

            public function enumerateObject():void
            {
```

```
                var o:Object = new Object();
                o.firstname = "MIcheal";
                o.lastname = "nabil";
                o.emailaddress = " MIcheal_nabil@hotmail.com
";

                for (var p:String in o)
                {
                    trace(p + ": " + o[p]);
                }
            }

        ]]>
    </mx:Script>

    <mx:Button label="Enumerate Object"
click="enumerateObject()" />

</mx:Application>
```

```
firstname: MIchal
emailaddress: MIcheal_nabil@hotmail.com
lastname: nabil
```

<u>*جملة التكرار The while Loop*</u>

فى هذه الطريقة يستمر بتنفيذ ما بداخل جملة التكرار ما دام الشرط متحقق فى كل مرة تريد فيها الدخول سوف يتحقق من الشرط اولا فاذا تحقق تقوم بالدخول الى داخل الجملة و تنفذ ما بداخلها الى ان يفشل و يخرج من جملة التكرار و لكن اذا كان هناك اوامر خارج جملة التكرار اى بعد جملة while سوف ينفذها .

لكن لاحظ ان جملة (while) يجب ان تحتوى على ما يلى:

١- متغير نضعه بالشرط لكى نتحقق من صحة الشرط .

٢- يجب وضع قيمة ابتدائية لهذا المتغير قبل جملة الـ(while) .

٣- يجب ان نذكر هذا المتغير و مقدار زيادته بداخل جملة الشرط سواءا قبل تنفيذ الجملة التى بداخل جملة التكرار او بعدها .

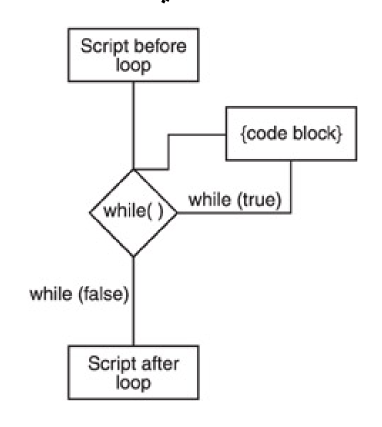
مثال :-

```
x = 0;
while( x < 100 ){
   trace(myNumber);
   x++;
}
```

** الجدول التالى يوضح صيغ عمليات التكرار المختلفة وتفسيرها ....

الـصـيـغـة	*الـتـفـسـيـر*
while	امر اللغة لعملية التكرار
)	قوس مفتوح يوضح بداخله بارامترات الامر
X<100	المتغير مع شرط لنهاية الحلقة
)	قوس نهاية بارامترات الامر – لاتضع بعده فاصلة منقوطة
{	قوس بداية بلوك الاوامر المطلوب تكرارها
بداخل اقواس البلوك توضع الاوامر المطلوب تكرارها	
x++;	و لا تنسى عداد الزيادة او النقصان ليتحقق الشرط لانهاء تنفيذ الحلقة
}	قوس نهاية بلوك الاوامر المطلوب تكرارها

وبعد الاعلان عن المتغير X خلاصة شرح الامر السابق (أعد تنفيذ ما بداخل البلوك طالما X او المتغير اقل من ١٠٠ ثم اقواس بلوك تضع ماشئت بداخله من اوامر وقوس نهاية البلوك ويزيد معنا فقط عداد للمتغير ليزيده بالمقدار الذي تريده ويوضع في اي مكان داخل الحلقة او حسب افكارك عن البرنامج المهم لاتنساه والا سوف يدور البرنامج داخل الحلقة الى الابد حيث ان شرط نهايتها ان تزيد X عن ١٠٠ وطالما لم تضع عداد زياده للمتغير فلن يتحقق الشرط وبالتالى لن تنتهي الحلقة الى الابد ويظل يعمل الكومبيوتر بلا نهاية للبرنامج ويميز العداد بالخلفية الصفراء واليك جدول صياغة الامر الذي تعودنا عليه.



The do while Loop حلقة التكرار

فى هذه الحالة يستمر تنفيذ ما بداخل الحلقة ما دام الشرط متحقق و هنا سوف يدخل الى داخل الحلقة و من ثم ينفذ الامر الذى بداخلها و بعد تنفيذها ينتقل ليتحقق من الشرط فاذا تحقق يعود مرة اخرى و اذا لم يتحقق يخرج من حلقة التكرار و لن يعود لها .

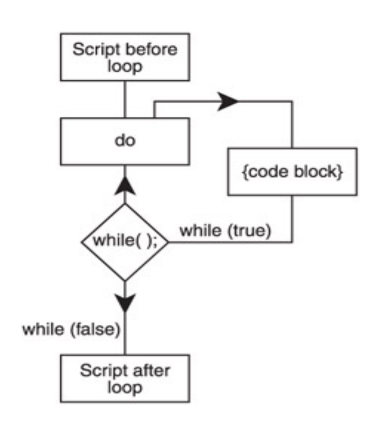
لكن لاحظ ان جملة (do while) يجب ان تحتوى على ما يلى:

١- متغير نضعه بالشرط لكى نتحقق من صحة الشرط .

٢- يجب وضع قيمة ابتدائية لهذا المتغير قبل جملة الـ(do while) .

٣- يجب ان نذكر هذا المتغير و مقدار زيادته بداخل حلقة الشرط سواءا قبل تنفيذ الجملة التى بداخل حلقة التكرار او بعدها .

```
myNumber = 99;
do{
   trace(myNumber);
}while(myNumber++ < 10);
```



قد نتسائل هنا ما الفرق بين while و جملة do while ؟؟؟؟؟

فى (while) نتحقق من الشرط قبل الدخول الى الحلقة اى اننا لا ننفذ اى شئ بداخلها ما دام الشرط لم يتحقق

و هذا امر طبيعى لاننا لم ندخل الى الحلقة اصلا فكيف نعرف ما بداخلها و ننفذه أما فى (do while) كنا ندخل الى الحلقة و ننفذ امر ثم نفحص الشرط و لكن بعد ان نكون قد نفذنا هذا الامر و يجب التنبيه هنا فى حالة عدم تحقق الشرط لن نعود مرة اخرى الى الـ(do) اذن الفرق هو ان بالـ (do while) ينفذ على الاقل امر واحد فى داخل حلقة التكرار حتى لو كان الشرط غير متحقق على العكس الـ(while) الذى لا ينفذ اى امر مادام الشرط غير متحقق .

Nested Loopsتداخل التكرار

```
var i:Number = 0;
while (++i <= 10) {
  var j:Number = 0;
  while (++j <= 10) {
    // perform these actions
  }
}
```

جـمـل الـتـحـكـّم فـي الـمـسـار Flow- control statement

إنّ البرمجة أعمق من أن تكون مجرّد تعريف متغيّرات.. إنّها تفكير منطقيّ يعتمد على حساب كل الاحتمالات، لاتخاذ الأفعال المناسبة لكل احتمال.. لهذا فلا بد أن توجد طرق نتحكّم بها فيما ينفّذ ومتى ينفّذ من البرنامج.

جملة الشرط ...If:

تستطيع أن تختبر حدوث شرط معيّن، فإذا كان صحيحا يتمّ تنفيذ مقطع الشرط، وإن كان خاطئا يقفز التنفيذ إلى جملة نهاية الشرط .

ولكن ما هو الشرط؟؟؟؟؟؟؟؟؟؟؟؟؟؟

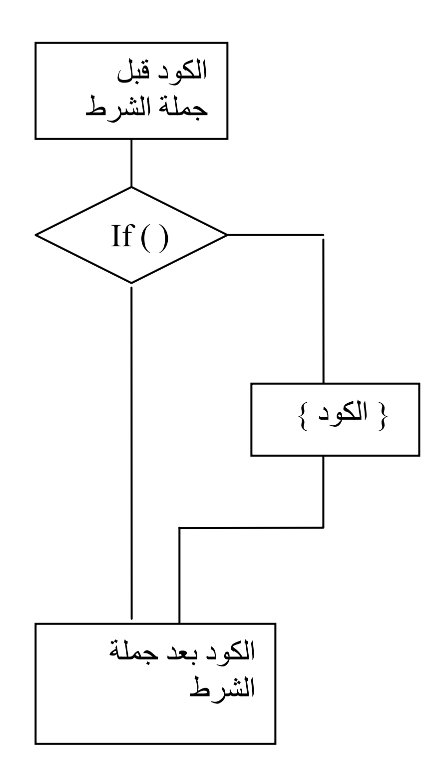
الشرط هو افتراض معين يتوقف عليه عمليات اخرى فمثلاً تريد ان تضع شرط الا يدخل رقم موظف اكبر من الف لان عدد موظفين الشركة لايزيدون عن الف وبالتالي اذا ادخل من يعمل على البرنامج رقم موظف اكبر من الف

يقوم البرنامج باصدار رسالة تفيد بذلك و هكذا لها حالات كثيرة حسب فكرة البرنامج

المهم انه تعبير يعطي نتيجة منطقيّة (True أو False)، مثل:

```
myNumber = 10;
if(myNumber < 20){
   trace("myNumber is less than 20");
}
```

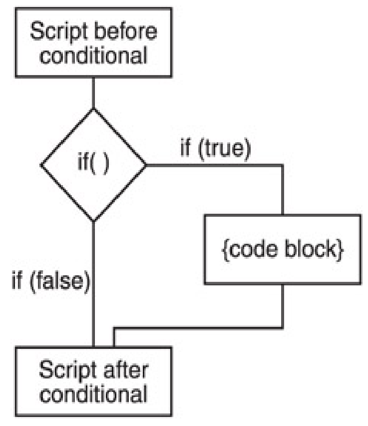
شرح المثال : لقد قمنا بتعريف متغير myNumber و اسندنا له القيمة ١٠ و قمنا بأختبار قيمة هذا المتغير اذا كانت اقل من ٢٠ فسوف نقوم بتنفيذ الكود الموجود بين القوسين { } و هو جملة trace اما اذا كانت قيمة المتغير اكبر من ٢٠ فلن يحدث شيء من الكود الموجود بين الاقواس { }



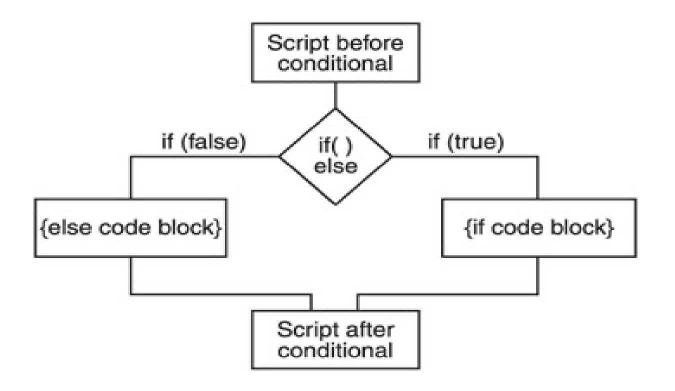
جملة الشرط If...else:

مثال :

```
myNumber = 10;
if(myNumber > 20){
   trace("myNumber is greater than 20);
}
Else{
   trace("myNumber is less than or equal to 20);
}
```



شرح المثال : لقد قمنا بتعريف متغير myNumber و اسندنا له القيمة ١٠ و قمنا بأختبار قيمة هذا المتغير اذا كانت اكبر من ٢٠ فسوف نقوم بتنفيذ الكود الموجود بين القوسين { } و هو جملة trace اما اذاقمنا بتغيير قيمة المتغيرلتصبح ٥٠ فسيتم تنفيذ الكود الموجود بعد كلمة else



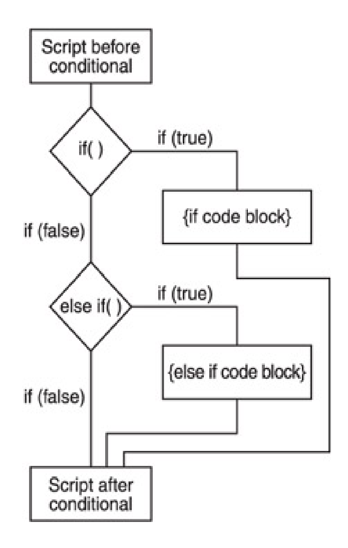
جملة الشرط if...Else :-

يمكن دمج جملتى if من خلال استخدام التعبير else if

```
myNumber = 10;
if(myNumber < 20){
trace("myNumber is less than 20");
}

else if(myNumber < 50){
trace("myNumber is less than 50 but greater than or equal to 20");
```

```
}
```



استخدام if مع معاملات اخرى مثل and او or:-

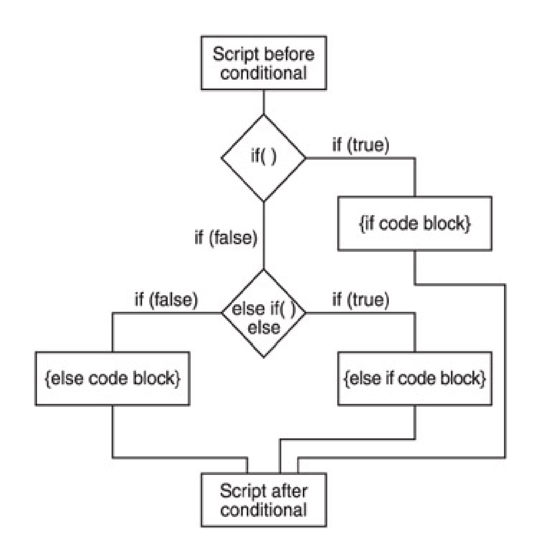
```
on (press) {
   if ((a == 7) and (b == 15)) {
    gotoAndPlay(20);
}
}
```

مثال اخر:-

```
on (press) {
   if ((a == 7) or (b == 15)) {
    gotoAndPlay(20);
}
}
```

كما يمكن دمج اكثر من تعبير فى جملة if كما يلى :-

```
myNumber = 10;
if(myNumber < 20){
    trace("myNumber is less than 20");
}
else if(myNumber < 50){
    trace("myNumber is less than 50 but greater than or equal to
20");
}
else{
    trace("myNumber is greater than or equal to 50");
}
```



<u>*الدالة الشرطية جملة Case .... switch:-*</u>

عندما يكون لدينا عدة خيارات و نكون نريد أن نخرج بواحد منهم و هو الذى نريده من بين الخيارات والذى سوف نخرج به سوف يحدده المتغير الذى سوف ندخله و الذى سوف يتفق مع واحدة من هذه الخيارات و يحققه ..

```
switch (المتغير){
   case الاحتمال الاول:
      المطلوب لهذا الاحتمال
      case الاحتمال الثانى:
      المطلوب لهذا الاحتمال
في حالة عدم تطابق اى حالة يتم تنفيذ ال
   ...
   [default]
}
```

The switch Statement

```
switch (expression){
   case caseClause1:
      code block
   case caseClause2:
      code block
   ...
   [default]
}
```

مثال:-

فى المثال التالى عرفنا متغير x و قيمته ١٠ وهناك حالتين لهما نفس قيمة شرط الحالة و هو ١٠ لذلك سوف تجد الناتج دائما هو الحالة الاولى case 1 و لن يطبع ابدا الحالة الثانية.

```
x = 10;
switch( x ){
   case 10:
      trace("case 1");
      break;
   case 10:
      trace("case 2");
      break;
}
```

مثال اخر:-

```
exp = "hello";
switch( exp ){
   case "hello":
      trace("case 1");
   case "hi":
      trace("case 2");
      break;
}
```

الناتج

case 1

مثال أخر يوضح التشابة بين فكرة استخدام if للتحكم فى البرنامج و استخدام switch :-

```
switch( exp ){
   case 1:
      //do task a
   case 2:
      //do task b
   case 3:
      //do task c
}
```

يمكن عمل نفس الكود السابق باستخدام if هكذا :-

```
if( x == 3 ){
   //do task c
}
else if( x == 2 ){
   //do task b
   //do task c
}
```

```
else if( x == 1 ){
   //do task a
   //do task b
   //do task c
}
```

مثال اخير يوضح جملة switch :-

```
switch( user_command_string ){
   case "move north":
   case "go north":
   case "north":
   case "n":
      trace("you have moved north");
      break;
   case "move south":
   case "go south":
   case "south":
   case "s":
      trace("you have moved south");
      break;
   default:
      trace("I'm sorry, I don't understand "+ user_command_string);
}
```

الخلاصة : اننا ندخل المتغير فى جملة (switch) للمقارنة مع الخيارات الموجودة بداخلها و عند مطابقة المتغير مع احدى الخيارات تصبح النتيجة (true) وسوف تنفذ الجملة المتعلقة بهذه المطابقة مع العلم ان واحد فقط من هذه الخيارات يعطى (true) و البقية (false) و عندما تكون كل الخيارات ليست مطابقة سوف ينفذ ما بداخل الـ(default) .

ما هي المصفوفة array

البرمجة الحقيقيّة تبدأ من هذه النقطة، فلقد صُنع الكمبيوتر أساسا، ليقوم بالعمليات الرتيبة المتكرّرة لآلاف أو ملايين المرّات، بسرعةٍ وبدون ملل.

افترض مثلا أنّك تريد حساب متوسط العمر لخمسين طالبا.. أوّل ما ستفكّر فيه، هو أن تعرّف خمسين متغيّرا وتجمعها معا وتقسم الناتج على ٥٠.. إنّ مثل هذه الطريقة ستستهلك منك شهرا على الأقلّ لكتابتها، وهي كفيلة بجعلك تكره البرمجة اساسا!

مع أنّك تستطيع أن تكتب هذا البرنامج في خمس سطور لا غير.. تخيّل!

والفكرة كلّها تعتمد على تخزين أعمار الطلبة في "تركيب ما"، يمكن للكمبيوتر أن يتعامل معه بطريقة آليّة رتيبة متكرّرة، لينفذ عليه العمليّات التي نريدها.

هذا التركيب هو المصفوفة Array، التي تتكوّن من مجموعة من الخانات، كل خانة منها تحتفظ بقيمة معيّنة، بحيث يمكن الوصول لهذه القيمة عن طريق رقم خانتها Index.

```
var array:Array = new Array();
```

```
var array:Array = new Array(elements);

var array:Array = new Array(element0,...elementN);
var letters:Array = ["a", "b", "c"];
items[4] = "apples";
trace(items[4]);
```

لاحظ ان اكشن اسكربت لا تهتم بنوع القيم الموجودة داخل المصفوفة
مثال هذة المصفوفة تحتوى على انواع مختلفة من المتغيرات

string, integer, a Boolean, and an object:

```
var data:Array = ["a", 2, true, new Object()];
```

اضافة عناصر لبداية المصفوفة و نهايتها

```
var array:Array = new Array();
array.push("val 1", "val 2");
```

و يمكن اضافة عنصر اخر من خلال معرفة طول المصفوفة

```
array[array.length] = "val 3";
```

مثال عمل مصفوفة مكونة من اربعة عناصر و اضافة عنصر جديد z فى بداية المصفوفة لذلك القيمة a سوف تتحرك من بداية المصفوفة اى من الترتيب (٠) داخل فهرس المصفوفة الى الترتيب (١) و هكذا

```
var letters:Array = new Array( );
letters.push("a", "b", "c", "d");
letters.unshift("z");
```

و لعرض عناصر المصفوفة نقوم بعمل جملة تكرار بالكود التالى :

```
for (var i:int = 0; i < letters.length; i++) {
  trace(letters[i]);
}
```

عرض محتويات مصفوفة من خلال جملة for

```
var letters:Array = ["a", "b", "c"];
for (var i:int = 0; i < letters.length; i++) {
    trace("Element " + i + ": " + letters[i]);
}
```

عرض محتويات المصفوفة بالعكس :

```
var letters:Array = ["a", "b", "c"];
for (var i:int = letters.length - 1; i >= 0; i--){
    trace("Element " + i + ": " + letters[i]);
}
```

يمكنك استخدام طول المصفوفة باسناد القيمة الى متغير مثال :

```
var length:int = sprites.length;
for (var i:int = 0; i < length; i++){
   sprites[i].x++;
}
```

البحث عن قيمة عنصر معين داخل المصفوفة :

```
// قم بعمل مصفوفة مكونة من ٨ عناصر
var letters:Array = ["a", "b", "c", "d", "a", "b", "c", "d"];

// حدد القيمة المراد البحث عنها و ضعها فى متغير نصى
var match:String = "b";

// قم بعمل جملة for لعرض محتويات المصفوفة
for (var i:int = 0; i < letters.length; i++) {

   // تأكد من ان القيمة المراد البحث عنها متتطابقة ام لا
   if (letters[i] == match) {

      اذا وجدت العنصر المراد البحث عنة اظهر رسالة
      trace("Element with index " + i +
            " found to match " + match);

// استخدم كلمة break للخروج من جملة التكرار
      break;
   }
}
```

قم بتجربة المثال التالى :

```
var letters:Array = ["a", "b", "c", "d", "a", "b", "c", "d"];

var match:String = "b";

// استخدم جملة التكرار للبحث
// the "b" is at index 5.
for (var i:int = letters.length - 1; i >= 0; i--) {
   if (letters[i] == match) {
      trace("Element with index " + i +
            " found to match " + match);
      break;
   }
}
```

يمكنك تبسيط عملية البحث من خلال استخدام الكلاس

//The class is in the *ascb.util* package

```
import ascb.util.ArrayUtilities;

var letters:Array = ["a", "b", "c", "d"];
```

```
trace(ArrayUtilities.findMatchIndex(letters, "b"));
// Displays: 1

trace(ArrayUtilities.findMatchIndex(letters, "r"));
// Displays: -1
```

ملحوظة كلاس المصفوفة بها ٣ وظائف
findMatchIndex( ), findLastMatchIndex( ), and findMatchIndices( ).

مثال :

```
public static function findMatchIndex(array:Array,
element:Object):int {
   // استخدم متغير لتحديد بداية الفهرس //
   // اختر ان تبدأ من الصفر //
   var startingIndex:int = 0;


   var partialMatch:Boolean = false;

   if(typeof arguments[2] == "number") {
      startingIndex = arguments[2];
   }
   else if(typeof arguments[3] == "number") {
      startingIndex = arguments[3];
   }

   if(typeof arguments[2] == "boolean") {
      partialMatch = arguments[2];
   }

   var match:Boolean = false;

   for(var i:int = startingIndex;
         i < array.length; i++) {
      if(partialMatch) {
         match = (array[i].indexOf(element) != -1);
      }
      else {
         match = (array[i] == element);
      }
```

```
      if(match) {
        return i;
      }
    }

    return -1;
}
```

ازالة عنصر من المصفوفة :

لازالة عنصر من منتصف المصفوفة استخدم الوظيفة *splice( )*

لازالة عنصر من نهاية المصفوفة استخدم الوظيفة pop( )

لازالة عنصر من بداية المصفوفة shift( )

مثال :

```
var letters:Array = ["a", "b", "c", "d"];

// احذف العنصر الاول و اطبع قيمتة //
trace(letters.shift(  ));

// احذف العنصر الاخير و اطبع قيمتة.
trace(letters.pop(  ));

// The array has two elements left: "b" and "c".
for (var i = 0; i < letters.length; i++) {
   trace(letters[i]);
}
```

عندما تُزيلُ عنصر من المصفوفةّ ، تَحتاجُ لتَغيير قيمةِ متغيّر الفهرس وفقاً لذلك. يُصوّرُ المثالُ التاليُ ما يَحْدثُ إذا لم تُجدّدُ قيمة متغيّر الفهرسَ:

```
var numbers:Array = new Array(4, 10);
numbers[4] = 1;
trace(numbers);  // Displays: 4,10,undefined,undefined,1
for(var i:int = 0; i < numbers.length; i++) {
   if(numbers[i] == undefined) {
      numbers.splice(i, 1);
   }
}
trace(numbers);  // Displays: 4,10,undefined,1
```

لكن الطريقة الصحيحة كما يلي

```
var numbers:Array = new Array(4, 10);
numbers[4] = 1;
trace(numbers);  // Displays: 4,10,undefined,undefined,1
for(var i:int = 0; i < numbers.length; i++) {
  if(numbers[i] == undefined) {
```

```
        numbers.splice(i, 1);
        i--;
      }
    }
    trace(numbers);  // Displays: 4,10,1
```

يمكنك ايضا حذف عنصر فى مصفوفة و اضافة اخر فى نفس الوقت كما يلى :

```
var letters:Array = ["a", "b", "c", "d"];

// حذف عنصرين و اضافة ثلاثة عناصر بدلا منهم
// into letters starting at index 1.
letters.splice(1, 2, "r", "s", "t");

// المصفوفة الان تحتوى على ٥ عناصر:
// "a", "r", "s", "t", and "d".
for (var i:int = 0; i < letters.length; i++) {
   trace(letters[i]);
}
```

تحويل متغير نصى الى مصفوفة :

```
var list:String = "Peter Piper picked a peck of pickled peppers";
// استخدم المسافة للفصل بين الكلمات فى السطر السابق و وضع كل كلمة كعنصر فى المصفوفة
var words:Array = list.split(" ");
```

للتحويل من مصفوفة الى متغير نصى :

```
var letters:Array = ["a", "b", "c"];
trace(letters.join("|"));   // Displays: a|b|c
```

مثال اخر :

```
var letters:Array = ["a", "b", "c"];
trace(letters.join());   // Displays: a,b,c
```

ترتيب المصفوفة :

```
var words:Array = ["tricycle", "relative", "aardvark", "jargon"];
words.sort(  );
trace(words); // Displays: aardvark,jargon,relative,tricycle
var scores:Array = [10, 2, 14, 5, 8, 20, 19, 6];
scores.sort(  );
trace(scores);   // Displays: 10,14,19,2,20,5,6,8
```

لعمل ترتيب عددى

```
var scores:Array = [10, 2, 14, 5, 8, 20, 19, 6];
```

```
scores.sort(Array.NUMERIC);
trace(scores);   // Displays: 2,5,6,8,10,14,19,20
```

فى بعض الاحيان تود ترتيب المصفوفة فى حالة اذا كانت تحتوى على عناصر فريدة غير متكررة لذلك يمكنك ان تستخدم *Array.UNIQUESORT*

مثال :

```
var ranking:Array = [2,5,6,3,1,1,4,8,7,10,9];
var sortedRanking:Object = ranking.sort(Array.UNIQUESORT);
trace(sortedRanking);   // Displays: 0
trace(ranking);  // Displays: 2,5,6,3,1,1,4,8,7,10,9
```

مثال على عمل تغيير عشوائي للترتيب :

```
var numbers:Array = new Array( );
for(var i:int=0;i<20;i++) {
   numbers[i] = i;
}
numbers.sort(randomSort);
for(var i:int=0;i<numbers.length;i++) {
   trace(numbers[i]);
}
```

عمل ترتيب خاص

```
var bands:Array = ["The Clash",
                   "The Who",
                   "Led Zeppelin",
                   "The Beatles",
                   "Aerosmith",
                   "Cream"];
bands.sort( );
for(var i:int = 0; i < bands.length; i++) {
   trace(bands[i]);

   /* output:
      Aerosmith
      Cream
      Led Zeppelin
      The Beatles
      The Clash
      The Who
   */
}
```

لعمل هذا يجب استخدام sort() مع معاملات خاصة فى الدالة *bandNameSort التى سوف تقوم بعملها كما يلى*

```
var bands:Array = ["The Clash",
                   "The Who",
```

```
                "Led Zeppelin",
                "The Beatles",
                "Aerosmith",
                "Cream"];
bands.sort(bandNameSort);
for(var i:int = 0; i < bands.length; i++) {
    trace(bands[i]);
    /* output:
       Aerosmith
       The Beatles
       The Clash
       Cream
       Led Zeppelin
       The Who
    */
}

function bandNameSort(band1:String, band2:String):int
{
    band1 = band1.toLowerCase(  );
    band2 = band2.toLowerCase(  );
    if(band1.substr(0, 4) == "the ") {
        band1 = band1.substr(4);
    }
    if(band2.substr(0, 4) == "the ") {
        band2 = band2.substr(4);
    }
    if(band1 < band2) {
        return -1;
    }
    else {
        return 1;
    }
}
```

الدالة *bandNameSort( )* في البداية تقوم بتحويل الحروف الكبيرة الى حروف صغيرة ثم تتأكد من اذا كانت الكلمة تبدأ بكلمة the و مسافة فاذا تحقق الشرط تتخطى الاربع حروف الاولى ثم اخيرا نقوم بمقارنة الحروف

لعمل مصفوفة ثنائية الابعاد

```
var cars:Array = new Array();
cars.push(["maroon", 1997, "Honda"]);
cars.push(["beige", 2000, "Chrysler"]);
cars.push(["blue", 1985, "Mercedes"]);
```

```
cars.push(["gray", 1983, "Fiat"]);

for (var i:int = 0; i < cars.length; i++) {
   // The output is the same as in the
   // earlier parallel arrays example:
   // A maroon 1997 Honda
   // A beige 2000 Chrysler
   // A blue 1985 Mercedes
   // A gray 1983 Fiat

   TRace("A " + cars[i][0] + " " +
            cars[i][1] + " " +
            cars[i][2]);
}
```

طريقة اخرى لعرض محتويات المصفوفة Cars و لكنها ليست منظمة كالاولى

```
for (var i:int = 0; i < cars.length; i++) {
   for (var j:int = 0; j < cars[i].length; j++) {
      TRace("Element [" + i + "][" + j + "] contains: " +
          cars[i][j]);
   }
}
```

المقارنة بين مصفوفتين

```
var letters1:Array = ["a", "b", "c", "d"];
var letters2:Array = ["a", "b", "c", "d"];
trace(letters1 == letters2];  // Displays: false
```

مثال بطريقة اخرى من خلال فحص كل مصفوفة بأداة التكرار for

```
var equivalent:Boolean = true;
for(var i:int = 0; i < letters1.length; i++) {
   if(letters1[i] != letters2[i]) {
      equivalent = false;
      break;
   }
}
trace(equivalent);  // Displays: true
```

المقارنة باستخدام دالة جاهزة داخل الفلاش و هى *ArrayUtilities.equals( )*

```
var letters1:Array = ["a", "b", "c", "d"];
var letters2:Array = ["a", "b", "c", "d"];
trace(ArrayUtilities.equals(letters1, letters2));
// Displays: true
```

مثال على المقارنة بين مصفوفتين

```
public static function equals(arrayA:Array,
```

```
                        arrayB:Array,
                        bNotOrdered:Boolean):Boolean {
//  لأختبار اذا كانت كلا من المصفوفتان لهما نفس الطول //
    if(arrayA.length != arrayB.length) {
        return false;
    }

// عمل نسختين من المصفوفتان حتى لا تتاثر الاصل بالعمليات التى ستحدث //
    var arrayACopy:Array = arrayA.concat(  );
    var arrayBCopy:Array = arrayB.concat(  );

//  قم بترتيب المصفوفتان //
    if(bNotOrdered) {
        arrayACopy.sort(  );
        arrayBCopy.sort(  );
    }

    // بجملة التكرار قم بمقارنة عناصر كل مصفوفة بالاخرى //
    // false  اذا لم تتوافق العناصر مع بعضها قم بحذف المصفوفة و ارجع بالقيمة //
    for(var i:int = 0; i < arrayACopy.length; i++) {
        if(arrayACopy[i] != arrayBCopy[i]) {
            delete arrayACopy;
            delete arrayBCopy;
            return false;
        }
    }

    // true  اما اذا كانت متتطابقة احذف المصفوفتان و ارجع بالقيمة //
    delete arrayACopy;
    delete arrayBCopy;
    return true;
}
```

مثال على مصفوفة عناصرها ليست ارقام بل كلمات

```
var members:Object = new Object(  );
members.scribe = "Franklin";
members.chairperson = "Gina";
members.treasurer = "Sindhu";
trace(members.scribe); // Displays: Franklin
```

قراءة اسم عنصر فى المصفوفة السابقة

```
var members:Object = new Object(  );
members.scribe = "Franklin";
members.chairperson = "Gina";
members.treasurer = "Sindhu";
```

```
for (var sRole:String in members) {
    // Displays:
    // treasurer: Sindhu
    // chairperson: Gina
    // scribe: Franklin
    trace(sRole + ": " + members[sRole]);
}
```

الوراثة Inheritance

يمكنك عمل خلية جديدة مشتقة من خلية قديمة ترث الخلية الجديدة كل صفات الخلية القديمة و يزيد عليها بعض الخصائص و الدوال الخاصة بالخلية الجديدة و تسمى الخلية المشتقة الجديدة Subclass و لعمل ذلك يجب ان تستخدم كلمة extends يسبقها اسم الخلية الجديدة و يتبعها اسم الخلية القديمة و لتوضيح الفكرة افترض ان لديك خلية اسمها A و نريد ان نورث منها خلية جديدة تدعى B نستخدم الكود التالي :

```
package com.example {
import com.example.A;
public class B extends A {
}
}
```

- لاحظ انة لا يمكن الوراثة من اكثر من خلية فى نفس الوقت كما ان الخلية الجديدة مسموح لها فقط بالتعامل مع الخصائص العامة و المحمية public و protected لكن الخصائص الخاصة private غير مسموح التعامل معها و الخلايا فى نفس الحزمة البرمجية فقط يمكنها التعامل مع الخصائص الداخلية internal و لنفرض مثلا ان لدينا خليتين Aو B.

```
package com.example {
public class A {
private var _one:String;
protected var _two:String;
public function A( ) {
initialize( );
}
private function initialize( ):void {
_one = "one";
_two = "two";
}
public function run( ):void {
trace("A");
}
}
```

}

و فى هذا المثال B عرفت على انها خلية مشتقة من A و الخلية B تسطيع التعامل مع
_two and run( ),
و لايمكنها التعامل مع
_one or initialize( ).

تعددية التشكل Polymorphism

و تنقسم الى :

١- ***Over loading تعدد اشكال الدوال :***

هى امكانية تعريف الدالة الواحدة أى عدد من المرات بحيث تحمل فى كل مرة نفس الاسم و تختلف فى عدد المعاملات أو انواعها أو تختلف فى نوع القيمة المرتدة .
او بصيغة اخرى هى قدرة المبرمج على تعريف اكثر من دالة بنفس الاسم و لكن يشترط ان تختلف كل من هذة الدوال فى قائمة ارسال و استقبال المعاملات أيا كان هذا الاختلاف فى العدد أو النوع أو كليهما .

٢- ***Overriding تعدد اشكال الخلايا :***

Overriding

هى امكانية ان نبنى خلية جديدة (تسمى الخلية الفرعية Subclass او الخلية المشتقة Derived Class) من الخلية الأب Parent Class او احيانا تسمى الخلية الاساسية Base class بحيث ترث كل خصائص و دوال الخلية الاب مع امكانية احتوائها على خصائص جديدة أو تعديل عمل بعض الخصائص القديمة باستبدالها بأخرى معدلة .
و لكى تقوم بعمل ذلك يجب ان تستخدم كلمة override قبل اسم الدالة المراد تعديلها فى الخلية القديمة مثال عمل override للدالة run:
keyword in the method declaration; the following overrides the run( ) method:

```
package com.example {
import com.example.A;
public class B extends A {
override public function run( ):void {
trace("B");
}
}
}
```

التعامل مع الاحداث Handling Events

الاحداث هى كل ما يستطيع الكائن الاستجابة له و يستجيب الكائن أما للأحداث التى يتسبب فيها المستخدم مثل الضغط على لوحة المفاتيح او النقر على زر الفأرة أو عامل وقتى مثل المؤقتات .timers

مثلا يمكنك عمل دالة لانتظار حدث النقر على زر فى النموذج فيظهر لة رسالة تحذيرية و لعمل ذلك انت تحتاج لعنصرين هما :

١- كائن لترتبط بة الحدث Event object من نفس خلية هذا الحدث Event class.

٢- دالة انتظار الحدث Event Listener .

بالنسبة للمثال السابق فأن كائن الحدث الخاص بحدث النقر على الزر يجب ان يكون مشتق من الخلية الخاصة بأحداث الفأرة Mouse Event Class. و هذا الكائن يمتلك دالة عامة هى addEventListener( ) التى تأخذ اسم الحدث و دالة انتظار الحدث كمعاملات لهذة الدالة و الصيغة العامة لذلك هى

```
object.addEventListener("eventName", listenerFunction);
```

و لتحويل المثال السابق لكود اكتب التالى:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute"
initialize="initializeHandler(event)">
<mx:Script>
<![CDATA[
import mx.controls.Alert;
private function initializeHandler(event:Event):void {
button.addEventListener(MouseEvent.CLICK, clickHandler);
}
private function clickHandler(event:MouseEvent):void {
Alert.show(event.toString( ));
}
]]>
</mx:Script>
<mx:Button id="button" />
</mx:Application>
```

لاحظ ان كل حدث ينتمى لخلية معينة فأحداث الفأرة تنتمى الى خلية Mouse Event و هذة الخلية تضم اشكال عديدة من الاحداث و لكل حدث اسم يعبر عنة بالطبع و علية نستنتج ان اسم الحدث ثابت لا يمكن تغييرة لذلك قام مصممى برنامج Flex2 باعتبار هذة الاحداث ثوابت Constant فمثلا CLICK ثابت و هناك ثوابت اخرى مثل :

DOUBLE_CLICK
MOUSE_DOWN
MOUSE_MOVE
MOUSE_OUT
MOUSE_OVER
MOUSE_UP
MOUSE_WHEEL
ROLL_OUT
ROLL_OVER

و ايضا لوحة المفاتيح لها أحداث تنتمى للخلية KeyboardEvent و الثابت الذى يمثل اسم حدث الضغط على لوحة المفاتيح هو KEY_DOWN .

- افترض الان انك تريد ازالة دالة انتظار الحث التى قمنا بعملها فى المثال السابق فماذا نعمل ؟
نقوم باستعمال الدالة removeEventListener للأزالة دالة الانتظار ولعمل ذلك فى المثال السابق كم بكتابة الكود التالى :

```
button.removeEventListener(MouseEvent.CLICK, onClick);
```

التعامل مع الاخطاء Error Handling

لغة الاكشن اسكربت تدعم التعامل مع اخطأ و قت التشغيل Runtime Error و ذلك يعنى انة يمكننا التعامل مع الاخطاء التى قد تحدث من برامجنا اثناء عملها من خلال توقعها و حلها .
و لغة الاكشن اسكربت تتعامل مع نوعين من الاخطاء هما :

1- Handling Synchronous Errors
2- Handling Asynchronous Errors

التعامل مع الاخطاء المتزامنة Handling Synchronous Errors

الاخطاء المتزامنة هى الاخطاء التى تحدث عند محاولة تنفيذ جملة برمجية .
و لحل هذة المشاكل قم باستخدام الدوال

try/catch/finally

مثلا اذا كان لديك جملة برمجية تتوقع منها مشاكل و قت التشغيل قم بكتابتها بين دالة try هكذا

```
try {
// الكود الذى من الممكن ان ينتج عن تفيذة خطأ{
```

ثم بعد ذلك تقوم باصطياد الخطأ من خلال كلمة Catch المعرف بها نوع الخطأ المرسل لها و بالطبع توقع الاخطاء التى قد تصدر من برنامج Flex2 نفسة اولا التى تكون محددة بارقام معينة و تنتمى للخلية flash.errors.Error ثم بعد ذلك اخطائك انت البرمجية الغير معروفة لبرنامج Flex2 مثال :

```
try {
// الكود الذى من المحتمل ان ينتج منة خطأ
}
catch (error:IOError) {
// الكود الذى سيحدث عندما يظهر خطأ معين{
catch (error:Error) {
// الكود الذى سيحدث عندما يحدث خطأ غير متوقع و لا تعرف رقم هذا الخطأ
}
```

و اخيرا يمكنك اضافة تعبير finally و تكتب فية الكود المراد تنفيذة على اية حالة فى حالة حدوث الخطأ عموما .

```
try {
// الكود الذى من المحتمل ان ينتج منة خطأ {
catch (error:IOError) {
// الكود الذى سيحدث عندما يظهر خطأ معين
}
catch (error:Error) {
```

```
} الكود الذى سيحدث عندما يحدث خطأ غير متوقع و لا تعرف رقم هذا الخطأ //
finally {
الكود المطلوب تنفيذة على اية حال //
}
```

عموما معظم اخطاء برنامج flex2 هى اخطاء متزامنة مثلا اذا كنت تريد برنامج يطلب من المستخدم ان يختار ملف ما من جهازة الشخصى فانك تقوم بعمل نافذة اختيار ملف و برنامج flash player لا يسمح بفتح اكثر من مربع حوار اختيار ملف واحد اى نافذة واحدة و اذا قام المستخدم باستدعاء الدالة browse الخاصة بالكائن FileReference اكثر من مرة فان برنامجك سيحدث بة خطأ تزامنى و لترى ذلك اكتب الكود التالى :

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute"
initialize="initializeHandler(event)">
<mx:Script>
<![CDATA[
import flash.net.FileReference;
private function initializeHandler(event:Event):void {
var file:FileReference = new FileReference( );
file.browse( );
file.browse( );
}
]]>
</mx:Script>
</mx:Application>
```

و لمعالجة رسالة الخطأ استخدم ما تعلمتة للأصطياد الاخطاء كما يلى

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute"
initialize="initializeHandler(event)">
<mx:Script>
<![CDATA[
import flash.net.FileReference;
private function initializeHandler(event:Event):void {
var file:FileReference = new FileReference( );
try {
file.browse( );
file.browse( );
}
catch(error:Error) {
errors.text += error + "\n";
}
}
```

```
]]>
</mx:Script>
<mx:TextArea id="errors" />
</mx:Application>
```

Handling Asynchronous Errors التعامل مع الاخطاء غير المتوقعة

مثلا اذا كنت تريد رفع ملف على الشبكة و هذا الملف اصبح غير موجود فان هناك خطأ غير متوقع و خارج نطاق اخطاء برنامج flex2 مثلا لحل المشكلة السابقة قم بكتابة الكود التالى

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute"
initialize="initializeHandler(event)">
<mx:Script>
<![CDATA[
private function initializeHandler(event:Event):void {
var loader:URLLoader = new URLLoader( );
// security sandbox  لتجربة ذلك يجب ان تختار ملف خارج نطاق
loader.load(new URLRequest("data.xml"));
loader.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
securityErrorHandler);
}
private function
securityErrorHandler(event:SecurityErrorEvent):void {
errors.text += event + "\n";
}
]]>
</mx:Script>
<mx:TextArea id="errors" />
</mx:Application>
```

استعمال لغة XML

إن XML هي طريقة لتمثيل أى بيانات منظّمة، وذلك بتحويلها لنص يعبر عنها.. وبهذا تصلح لغة XML للتعبير عن أى نوع من أنواع البيانات، مثل الجداول والصور وغيرهما.
ورغم أنّ الملفّات النصية أكبر حجما من الملفّات الثنائيّة Binary Files، إلا إنّ الأولى صالحة للتعامل مع أيّ تطبيق بل مع أيّ نظام تشغيل.. لهذا فقد صارت لغة XML في السنوات الأخيرة هي أنسب وسيلة لنقل البيانات عبر الإنترنت، وذلك حتّى تتجاوز مشاكل عدم التوافق بين التطبيقات وأنظمة التشغيل المختلفة..

لغة الترميز الموسعة eXtensible Markup Language التي يرمز لها بالاختصار XML وهى تستخدم في وصف وتخزين وتنظيم البيانات بخلاف لغة HTML التي تستخدم لكيفية عرض البيانات على المتصفح .

و هنا يجب ملاحظة شي هام أن لغة XML ليست لغة في الأصل فا XML تصف مجموعة من التعابير التي تستخدمها لبناء لغاتك الخاصة على سبيل المثال لنفترض أن لدينا بيانات حول اسم شخص ما وانك تريد تبادل هذه البيانات مع الآخرين يمكنك تمثيل هذه البيانات في ملف نصي بالصورة التالية.

Michael nabil

أو بصيغة HTML بهذا الشكل .

```
<html>

<head><title>name></title></head>

<body>

<p> Michael nabil </p>

</body>

</html>
```

يمكن تمثيل هذه البيانات في XML بالشكل التالي .

```
<name>

<first> Michael</first>

<last> nabil </last>

</name>
```

يمكنها بسهولة معرفة أن هذه المعلومات تمثل اسم Name لشخص ما وأيضا هناك بيانات تسمى <first>و بيانات أخرى تسمى <last> يجب آن تكون ذات معنى طبعا المعنى يدل على محتوى المعلومة بداخلها .

لو قمنا بحفظ الملف السابق باسم name.xml فيمكنا فتح هذا الملف بواسطة متصفح الإنترنت لديك بشرط أن لا يقل عن ٥,٥ وسوف يظهر بهذا الشكل .

```
- <name>
    <first>Michael</first>
    <last>nabil</last>
  </name>
```

وبالرغم من أن ملف XML السابق لا يحتوى على إي معلومات حول كيفية العرض فان المتصفح قام باستعراض الملف بصورة لطيفة وبتنسيق لونى مختلف وأيضا البنية الشجرية التي فهمها المتصفح وترجمها أيضا وذلك بالنقر على الرمز (-) بجانب البند <name> وهذه الطريقة مفيدة جدا عندما يكون الملف كبير الحجم .

ونلاحظ أننا لم نقم بوصف البيانات لكي تظهر بهذا التنسيق الموجود ولكن هذا ما يقدمه لنا متصفح الإنترنت فلهذا المتصفح ورقة تنسيق Style Sheet افتراضية مبيتة داخلة مما يمكن المتصفح من عرض إي مستند XML وفق ورقة التنسيق هذه .

تدخل لغة CSSضمن عائلة XML أيضا وذلك في حالات استعراض مستندات XML البسيطة ويمكن العوض عنها باستخدام لغة XSL وذلك في الحالات المعقدة وهى تتضمن تحويلات خاصة تسمى هذه التحويلات XSLT والتي تستخدم لتحويل مستندات XML إلى أنواع مستندات أخرى .بإضافة إلى أسلوب عرض المعلومات .

نموذج كائن المستند Document Object Model أو DOM هذا النوع من عائلة XML هو معروف جيداً لمن تعامل من قبل مع لغة DHTML و JavaScript ألم تمر عليك جملة Document.write قبل ذلك .

فهذا الكائن يمكنه ربط مستندات XML مع لغات برمجة أخرى مع إمكانية الإضافة والحذف التعديل داخل مستندات XML بواسطة لغتك المفضلة

مطوري البرمجيات تستخدم هذا النموذج منذ سنوات عدة باستخدام بنية بينات تسمى Object model أو نموذج الكائن وهى مرتبطة مع بعضها بتسلسل هرمي . أيضا في لغة XML تجمع البينات في تسلسل هرمي فالبنود في المستند تتبع بعضها البعض بعلاقات Parent / Child أو الأب / الابن .

وهذه البنود تسمى بالعناصر elements وهى أجزاء منفردة من المعلومات .

نأخذ مثال الاسم السابق شرحه ونمثله بطريقة هرمية

نلاحظ أن البند <Name > هو أب للبند <First > والبند <First > هو ابن للبند < Name> والبنود <First > و<Middle > و <Last > جميعها انساب لبعضها البعض لان جميعهم أبناء للبند <Name >

ونلاحظ أيضا أن النص هو ابن للعنصر الذي ينتمي له فالنص Emad يمثل ابنا للبند < First> .تسمى هذه البنية من البينات بالشجرة Tree فكل جزى من الشجرة يحتوى على أبناء تسمى بالفروع Branches وجميع الأجزاء التي تحتوى على أبناء تسمى بالأوراق Leaves

<u>إذا نقول :</u>

Element Content

العنصر <Name > يعتبر element content لان هناك عناصر تنتمي له وليس مجرد نص فإنه يعتبر محتوى عنصر .

Simple Content

العنصر < First> و < Middle>و< Last> هي محتوى بسيط Simple Content لأنها تحتوى على نص فقط .

Mixed Content

أيضا يمكن للعناصر أن تحتوى على عناصر أخرى وعلى نصوص في تلك الحالة فان للعناصر تلك محتوى مختلط Mixed Content

منهجية عمل XML وقواعدها :-

١) اللواحق والنصوص والعناصر
Tags and Text and Elements

اللاحقة أو ما يطلق عليها البعض والوسم (Tag) هي عبارة عن كلمة أو مصطلح موضوع بين رمزي إحاطة < > يمثل رمزا معرفا لتنسيق ما وذلك في مستندات HTML بينما يمثل اسما لعنصر Elements في مستندات XML

مثل ....

```
- <name>
    <first>Michael</first>
    <last>nabil</last>
  </name>
```

وكما تلاحظ فان الـ Tag تأخذ الطابع الزوجي فكل لاحقة لها لاحقة مقابله لها تعرف الأولى لاحقة البداية Start Tag وتعرف الثانية بلاحقة النهاية End Tag . الاختلاف بين الاثنين هو أن لاحقة النهاية تحتوى على الرمز "/" .

في XML جميع المعلومات الموجودة بين لاحقة البداية ولاحقة النهاية نسمى بالعناصر وبالتالي فان Element .

< first> هي لاحقة بداية

< /first> هي لاحقة نهاية

< firest>michael< /first> هو عنصر

Element Content
النص الواقع بين لاحقة البداية ولاحقة النهاية يسمى بمحتوى العنصر Element content

PCDATA
المحتوى الواقع بين لاحقتين عبارة عن بيانات ويعرف في هذه الحالة ببيانات الرمز المعرب PCDATAوذلك إذا احتوى هذه العنصر على معلومات نصية مثل العنصر < middle> فهوPCDATA

Root Element
المستند ككل بدء باللاحقة < name> وانتهاء باللاحقة</name > فهو يمثل عنصر يحتوى على مجموعة من العناصر و هنا نطلق عليه عنصر الجذرRoot Element

قوانين العناصر

يجب على مستندات XML الخضوع لهذه القوانين كي تشكل فعليا مستندات XML محكمة الهيئة Well-formed XML Documents

- لكل لاحقة بداية لاحقة نهاية مماثلة لها .
- لا يمكن للواحق أن تتداخل .
- يحتوى مستند XML على عنصر جذر واحد فقط.
- لغة XML حساسة لحالة الحروف Case-Sensitive
- لغة XML لا تتجاهل المساحات الفارغة في مستنداتها .

أسماء العناصر :-

لغة XML توفر لك الحرية في تسمية العناصر فهي لا تحتوى على أسماء محجوزة كما في معظم اللغات ، فهي لديها مرونة كبيرة في اختيار الأسماء . ولكن يوجد مجموعة من القوانين التي يجب مراعاتها :-

- يمكن للأسماء أن تبدأ بأحرف لاتينية أو غير لاتينية أو أن تبدأ بالرمز underscore(_) ولكن لا يمكن أن تبدأ برقم أو بعلامة ترقيم .
- بعد الحرف الأول يمكن للأسماء أن تحتوى على أرقام بالإضافة إلى الرمزين " _ " و " . "
- لا يمكن للأسماء أن تحتوى على فراغات .

- لا يمكن للأسماء أن تحتوى على ":" فهو محجوز في XML
- لا يمكن للأسماء أن تبدأ بالأحرف XML سواء كانت بأحرف صغيرة أو كبيرة .
- لا يمكن أن يكون هناك فراغ بين قوس الإحاطة المفتوح < وبين اسم العنصر

٢) الصفات Attributes

أن مستندات XML يمكن أن تتضمن صفات أو سمات معينة attributes الصفات عبارة عن اسم معين تسند له قيمة معينة بحيث يرتبط ذلك الاسم وتلك القيمة بعنصر معين في مستند XML .

مثل

```
- <name nickname="mic">
    <first>Michael</first>
    <last>nabil</last>
  </name>
```

يجب أن تحتوى الصفات على قيم ويجب أن تكون هذه القيم واقعة بين علامتي اقتباس ولا يشترط أن تكون علامة الاقتباس مفردة أو مزدوجة .

يمكن للصفات أن تقدم بيانات وصفية Metadata والتي يمكن أن لا تكون وثيقة الصلة بمعظم التطبيقات التي تتعامل مع المستندات XML

على سبيل المثال إذا علمنا أن بعض التطبيقات يمكن أن تهتم بالاسم المستعار Nickname ولكن معظم التطبيقات لا تهتم بهذه المعلومات فان استخدام هذه المعلومات كصفة سيكون ذا معنى .

ان flash player يدعم طريقتين للتعامل مع لغة Xml هما
1- a legacy XMLDocument class (XML DOM)
2- new XML class that implements the ECMAScript for XML (E4X) standard.
شرح لغة ال Xml خارج نطاق هذا الكتاب لكن يمكنك الرجوع لكتابى السابق اكشن اسكربت ٢ من الصفر لان بة ملحق لتعليم لغة ال Xml و للأمانة العلمية
هذا الملحق منقول من موقع الموسوعة العربية للبرمجة

http://www.c4arab.com

للكاتب : أسماء المنقوش

و يمكنك الرجوع لأى موقع اخر على الانترنت لتعلم منة لغة ال XML

طريقة XML DOM

- بداية يجب ان تقوم بعمل كائن Xml و لديك طريقتين لعمل ذلك اما تستخدم xml literals او تستعمل xml constructor .

- xml literals يستعمل اذا كنت تريد كتابة بيانات ملف ال xml و توصيفة داخل الكود .

مثال

```
var xml:XML = <books>
<book>
<title>learn Flex 2</title>
<authors>
<author first="michael" last="akhnokh" />
<author first="nermen" last="fahim" />
</authors>
</book>
<book>
<title>ActionScript 3.0 from Zero</title>
<authors>
<author first="frank" last="nabil" />
<author first="mena" last="nabil" />
<author first="deana" last="welsson" />
</authors>
</book>
</books>;
```

قمت بكاتبة محتويات ملف ال xml داخل متغير للأستخدامة مباشرة من داخل الكود .

- اما اذا كنت تريد كتابة ملف منفصل عن الكود فانت فى هذة الحالة يجب ان تستخدم xml constructor فيقوم البرنامج بتحميل البيانات من الملف كانها نصوص String و يرسلها لل Xml constructor كالكود التالى

```
var xml:XML = new XML(loadedXMLData);
```

- لكن لاحظ ان كل نص يرسل يعتبر عقدة من العقد الخاصة بملف ال Xml و هنا يعتبر البرنامج المسافات الفارغة بين العقد داخل ملف ال Xml نصوص و يرسلها للبرنامج مما يسبب خطأ اثناء تنفيذ البرنامج لذلك يجب ان تستخدم جملة تجاهل المسافات الفارغة وهى

```
XML.ignoreWhitespace = true;
var xml:XML = new XML(loadedXMLData);
```

Reading XML Data القراءة من ملف الاكس ام ال

يمكنك استخدام الدالة children( ) التى تقوم بأعادة مصفوفة من الكائنات تحتوى على كافة العقد الابن الخاصة بالعقدة الاب مثال عرض العقدة الاولى من ملف Xml:

```
var bookNodes:XMLList = xml.children( );
trace(bookNodes[0].toXMLString( ));
```

و لمعرفة عدد العقد داخل الملف استخدم الدالة

```
length( )
```

مثال :

```
var bookNodes:XMLList = xml.children( );
for(var i:uint = 0; i < bookNodes.length( ); i++) {
trace(bookNodes[i].children()[0].toXMLString( ));
}
```

مثال : على استخدام الدالة parent()

```
trace(xml.children()[0].children()[0].parent().toXMLString( ));
```

مثال : على استخدام الدالة attributes( )

```
var author0:XML = xml.children()[0].children()[1].children( )[0];
var attributes:XMLList = author0.attributes( );
var attributeName:String;
for(var i:uint = 0; i < attributes.length( ); i++) {
attributeName = attributes[i].name( );
trace(attributeName + " " + author0.attribute(attributeName));
}
```

يعتبر الاسلوب السابق هو المفضل فى حالة اذا كنت لا تدرى طريقة بناء او اسلوب ملف ال xml الذى تعمل معة لكن من المؤكد انها طريقة صعبة نوعا ما اما اذا كنت تعرف التراكيب المستخدمة فى بيانات ملف ال Xml الخاص بك فانا انصحك باستعمال الطريقة التالية :

طريقة(E4X)

تعتبرE4X اسهل فى التعامل حيث يمكنك التعامل مع العقد الصغيرة داخل الملف مباشرة من خلال الاسم الخاص بكل عقدة مثلا تريد ان تتعامل مع العقدة الاولى فى الملف

```
trace(xml.book[0]);
```

مثلا لعرض الاسم الاول للكاتب الاول من الملف

```
trace(xml.book[0].authors.author[0].toXMLString( ));
```

مثلا لعرض اول صفة من صفات الكاتب الاول فى اول عقدة

```
trace(xml.book[0].authors.author[0].@first);
```

استخدام E4X filters

مثلا لعرض كل اسماء اباء المؤلفين للكتب الذين اسمهم الاخير nabil

```
var authors:XMLList = xml.book.authors.author.(@last == "nabil");
for(var i:uint = 0; i < authors.length( ); i++) {
trace(authors[i].parent().parent().toXMLString( ));
}
```

الكتابة و التعديل باستخدام كائن اكس ام ال Writing to and Editing XML Objects

باستخدام تقنية E4X يمكنك تعديل اسم الكتاب الاول الى Flex2 with mic

```
xml.book[0].title = " Flex2 with mic ";
```

مثلا تغير اسم المؤلف الثانى الى youssef

```
xml.book[0].authors.author[1].@first = " youssef ";
```

اذا كنت تريد اضافة بيانات جديدة استخدم دالة

```
appendChild(  );

xml.book[0].appendChild(<publisher>MIC</publisher>);
xml.book[1].appendChild(<publisher>MIC</publisher>);
```

اما للأضافة عقد قبل عقدة ما او بعد عقدة ما استخدم كلا من دالتى

insertChildBefore( ) and insertChildAfter( )

مثال :

```
xml.book[0].insertChildAfter(xml.book[0].authors,
<publicationDate>2008</
publicationDate>);
xml.book[1].insertChildAfter(xml.book[1].authors,
<publicationDate>2008</
publicationDate>);
```

يمكنك اضافة عقدة للملف ثم حذفها الملف بالكود التالى :

```
xml.book[0].authors.author[1] = <author first="michael"
middle="nabil" last="aknokh"
/>;
trace(xml.book[0].authors);
delete xml.book[0].authors.author[1].@middle;
trace(xml.book[0].authors);
```

Reflection

لغة الاكشن اسكربت تعدم هذة الخاصية من خلال استخدام دوال معينة من الحزمة البرمجية flash.utils package و هذة الدوال هى :

- getQualifiedClassName
- getQualifiedSuperclassName
- getDefinitionByName
- describeType

Getting the Class Name

مثال : على معرفة اسم الخلية

```
var loader:URLLoader = new URLLoader( );
var className:String = getQualifiedClassName(loader);
trace(className); // Displays flash.net.URLLoader
```

مثال : على استعادة الاسم الكامل للخلية

```
var loader:URLLoader = new URLLoader( );
var className:String = getQualifiedSuperclassName(loader);
trace(className); // Displays flash.events.EventDispatcher
```

Getting the Class by Name

مثال : يمكنك معرفة ارتباط المرجعى لخلية ثم بعد ذلك عمل كائن من هذة الخلية

```
var classReference:Class =
Class(getDefinitionByName("flash.net.URLLoader"));

var instance:Object = new classReference( );
```

كما يمكنك استعمال القيمة الراجعة

```
var loader:URLLoader = new URLLoader( );
var className:String = getQualifiedClassName(loader);
var classReference:Class =
Class(getDefinitionByName(className));
var instance:Object = new classReference( );
```

Class Introspection

يمكنك استخدام describeType( ) لمعرفة وصف كافة الاحداث الخاصة بكائن و الدوال العامة و الخصائص العامة لة ايضا :

مثال : لمعرفة كافة المعلومات عن خصائص و دوال و احداث الكائن URLLOADER

```
var loader:URLLoader = new URLLoader( );
var description:XML = describeType(loader);
trace(description);
```

الناتج هو

```
<type name="flash.net::URLLoader"
base="flash.events::EventDispatcher"
isDynamic="false" isFinal="false" isStatic="false">
<metadata name="Event">
<arg key="name" value="httpStatus"/>
<arg key="type" value="flash.events.HTTPStatusEvent"/>
</metadata>
<metadata name="Event">
<arg key="name" value="securityError"/>
<arg key="type" value="flash.events.SecurityErrorEvent"/>
</metadata>
<metadata name="Event">
<arg key="name" value="ioError"/>
<arg key="type" value="flash.events.IOErrorEvent"/>
</metadata>
<metadata name="Event">
<arg key="name" value="progress"/>
<arg key="type" value="flash.events.ProgressEvent"/>
</metadata>
<metadata name="Event">
<arg key="name" value="complete"/>
<arg key="type" value="flash.events.Event"/>
</metadata>
<metadata name="Event">
```

```
<arg key="name" value="open"/>
<arg key="type" value="flash.events.Event"/>
</metadata>
<extendsClass type="flash.events::EventDispatcher"/>
<extendsClass type="Object"/>
<implementsInterface type="flash.events::IEventDispatcher"/>
<constructor>
<parameter index="1" type="flash.net::URLRequest"
optional="true"/>
</constructor>
<variable name="bytesTotal" type="uint"/>
<variable name="data" type="*"/>
<method name="load" declaredBy="flash.net::URLLoader"
returnType="void">
<parameter index="1" type="flash.net::URLRequest"
optional="false"/>
</method>
<method name="close" declaredBy="flash.net::URLLoader"
returnType="void"/>
<variable name="dataFormat" type="String"/>
<variable name="bytesLoaded" type="uint"/>
<method name="dispatchEvent"
declaredBy="flash.events::EventDispatcher"
returnType="Boolean">
<parameter index="1" type="flash.events::Event" optional="false"/>
</method>
<method name="toString"
declaredBy="flash.events::EventDispatcher"
returnType="String"/>
<method name="willTrigger"
declaredBy="flash.events::EventDispatcher"
returnType="Boolean">
<parameter index="1" type="String" optional="false"/>
</method>
<method name="addEventListener"
declaredBy="flash.events::EventDispatcher"
returnType="void">
<parameter index="1" type="String" optional="false"/>
<parameter index="2" type="Function" optional="false"/>
<parameter index="3" type="Boolean" optional="true"/>
<parameter index="4" type="int" optional="true"/>
<parameter index="5" type="Boolean" optional="true"/>
</method>
```

```
<method name="hasEventListener"
declaredBy="flash.events::EventDispatcher"
returnType="Boolean">
<parameter index="1" type="String" optional="false"/>
</method>
<method name="removeEventListener"
declaredBy="flash.events::EventDispatcher"
returnType="void">
<parameter index="1" type="String" optional="false"/>
<parameter index="2" type="Function" optional="false"/>
<parameter index="3" type="Boolean" optional="true"/>
</method>
</type>
```

استخدام خاصية ربط البيانات Using Data Binding

هى طريقة لربط أداة بأداة اخرى او كائن تم عملة بلغة الاكشن اسكربت بأداة اخرى و فكرة الربط هى عندما تتغير قيمة معينة فى كائن او اداة فانها تؤثر فى الكائن الاخر و هناك اكثرمن طريقة لتحقيق ذلك و لكننا الان سنتعلم ابسط هذة الطرق و هى طريقة كتابة الاقواس
Curly braces ({})
التى بداخلها الاداة المراد الربط بينها و بين اداة اخرى و فى هذة الطريقة نكتب الكود فى داخل تاج MXML

مثال : نستخدم مربع نص و زر ادخال و كلا منهما يمتلك خاصية كتابة و قراءة النص المعروض من خلالهم و الان سنكتب كود يعرض كلمة Michael فى مربع النص

```
<mx:VBox>
<mx:Text id="output" text="Michael" width="200" height="200" />
<mx:TextInput id="input" />
</mx:VBox>
```

الان سنستخدم خاصية ربط البيانات بين مربع النص و الزر بحيث كلما يتغير النص المكتوب فى مربع النص يتغير النص المكتوب على الزر

```
<mx:VBox>
<mx:Text id="output" text="{input.text}" width="200" height="200"
/>
<mx:TextInput id="input" />
</mx:VBox>
```

الفرق بين كلمتى Import and Include

: تقوم بعمل ربط بين كلاس خارجى و بين المشروع الحالى لكى يسهل لك استخدام خصائص هذة الكلاس داخل ملف المشروع الجديد الخاص بك مثال Import

```
import mx.controls.TextInput;
```

: نستخدمها فى حالة نقل سطر من الكود من ملف اخر حتى لا نقوم بكتابة الكود اكثر من مرة Including.

و يمكن كتابة الكود بين تاجى <mx:Script> .

مثال :

```
<mx:Script>
   <![CDATA[
      include "filename.as";
   ]]>
</mx:Script>
```

او يمكن كتابة اسم الملف الموجود بة كود الاكشن اسكربت كمايلى

<mx:Script source="filename">

الفكرة من عمل components

بداية ما هى الكمبونات هى مجموعة من الأدوات التى نستخدمها فى برنامج Flex و لكن هذة المكونات صنعتها الشركة المصممة لبرنامج Flex و أعطتنا مطلق الحرية للأستخدامها كما هى و أيضا تعديلها سواء بالأضافة عليها أو صنع غيرها من جديد و لكن ما الغاية من فكرة الكومبونات :

١- تقسيم المشروع لنماذج صغيرة يمكن استخدامها منفردة فى برامج أخرى .
٢- تشارك المبادئ العامة لعمل الكمبونات بينها و بين بعضها .
٣- عمل مجموعة من الكمبيونات تساعدك فى مشاريعك القادمة و يعاد استخدامها او بيعها لمبرمجين اخرين .

مثال : يمكنك عمل كمبونات خاصة بك بالاعتماد على كمبونات سابقة و هى TextInput و سوف يتضح هذا عند شرح البرمجة بالكائنات و مفهوم الوراثة .

```
package myComponents
{
   public class MyTextInput extends TextInput
   {
      public function MyTextInput()
      {
         ...
      }
      ...
   }
}
```

Flex Class Library مكتبة الخلايا الخاصة بالبرنامج

مثال على مجموعة من الخلايا الهامة التى تشترك فى بناء برنامج Flex

- mx.controls - Flex user interface controls الخلايا الخاصة بواجهة المستخدم
- mx.collections - Flex collection components
- mx.charts - Flex charting components الخلايا الخاصة بالرسم البيانى
- mx.utils - Flex utility classes
- flash.events - Flex event classes الخلايا الخاصة بالاحداث
- flash.net - Flex classes for sending and receiving from the network, such as URL downloading and Flash Remoting
  - هى الخلايا الخاصة بربط البرنامج مع الشبكات عموما و شبكة الانترنت ايضا

مزود البيانات و مجموعات جمع البيانات Data Providers and Collections

الكثير من أدوات برنامج Flex تستخدم *data provider* *مثل* array or ArrayCollection و ذلك كمصدر يمد الاداة بالبيانات التى نريد عرضها من خلالها مثل اداة الداتا جريد Data grid تعرض البيانات فى شكل صفوف كبرنامج الاكسيل و كل صف يحصل على بياناتة من مصفوفة بها البيانات .

و فيما يلى مجموعة الادوات التى يمكن استخدام Data provider معها :

- Tree control
- DataGrid control
- ComboBox control
- List control
- TileList control
- Menu control
- MenuBar control
- ButtonBar control
- LinkBar control
- TabBar control
- ToggleButtonBar control
- LineChart control
- ColorPicker control
- PopUpMenuButton control
- HorizontalList control

- Repeater component
- DateField control

و اسهل طريقة للأستخدام data provider مع اداة هى تعريف مصفوفة من الكائنات او المتغيرات النصية التى يتم اسنادها الى خصائص dataProvider المرتبط بالاداة .

و فى حالة استخدام raw data object كمزود بيانات data provider يعتبر محدود الامكانيات بعكس استخدام collection classes المفيد جدا فى حالة تغير البيانات .

حيث Raw objects ليس لدية امكانية تحديث البيانات فى الاداة فى حالة حدوث تغيرات فى الكائن الاساسى .

المثال التالى يوضح الفرق فى حالة استخدام Array (raw object)

و فى حالة استخدام Array Collection

قم بعمل ٢ زر two Button controls الاول يستدعى الدالة addCountryToArray() و الاخر يستدعى الدالة addCountryToCollection() و كلتا الدالتين تأخذ معاملاتهم من مربع نص TextInput و يرسلها لمزودى البيانات data providers .

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
        <![CDATA[

            import mx.collections.ArrayCollection;

            [Bindable]
            public var myArray:Array = ["United States", "South
Africa", "United
Kingdom"];

            [Bindable]
            public var myCollection:ArrayCollection = new
ArrayCollection
(["United States", "South Africa", "United Kingdom"]);

            public function addCountryToArray(country:String):void
            {
                myArray.push(country);
            }
```

```
            public function
addCountryToCollection(country:String):void
            {
                myCollection.addItem(country);
            }

        ]]>
    </mx:Script>

    <mx:TextInput id="countryTextInput" text="Argentina" />
    <mx:Label text="Bound to Array (raw object)" />
    <mx:Button click="addCountryToArray(countryTextInput.text)"
label="Add Country
to Array" />
    <mx:List dataProvider="{myArray}" width="200" />

    <mx:Label text="Bound to Collection" />
    <mx:Button
click="addCountryToCollection(countryTextInput.text)" label="Add
Country to Collection" />
    <mx:List dataProvider="{myCollection}" width="200" />

</mx:Application>
```

عندما تضغط على زر the Button control labeled Add Country to Array ستجد ان

first List control البيانات بها لن تتغير بعكس الضغط على الزر الاخر .

<u>استخدام الاكشن اسكريبت في عمل اداة</u>

<u>Using ActionScript to Create Visual Components</u>

استخدم كلا من دالة

addChild() or addChildAt()

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    verticalAlign="middle"
    horizontalAlign="center"
    xmlns="*">

    <mx:Script>
```

```
        <![CDATA[

            import mx.controls.TextInput;

            public function createComponent():void
            {
                var t:TextInput = new TextInput();
                t.text = "This component was created at runtime";
                this.addChild(t);
            }

        ]]>
    </mx:Script>

    <mx:Button label="Create Component"
click="createComponent()" />

</mx:Application>
```

كما نعلن عن بدأ تنظيم الكورسات التالية ،*للحجز و الاستعلام* اتصل

م/ مايكل نبيل اخنوخ

٠١٠٣٥٤٦٦٠٩

دبلومة صيانة الكمبيوتر

سعر الدبلومة ١٠٠ جنية

- التعريف بمكونات الكمبيوتر.
- كيفية تجميع جهاز كمبيوتر كامل .
- شرح اعدادت البيوس Bios .
- طرق تقسيم القرص الصلب ببرامج مختلفة .
- تنزيل ويندوز windows xp .
- طرق تعريف الكروت .
- عمل شبكة منزلية .
- اصلاح الاعطال الشائعة .

دبلومة الويندوز و الأوفيس ٢٠٠٧

سعر الدبلومة ٢٠٠ جنية

- Windows xp
- Microsoft word 2007
- Microsoft Excel 2007
- Microsoft power point 2007
- Microsoft access 2007

دبلومة الجرافيك المستوى الاول

سعر الدبلومة ٣٥٠ جنية

- مبادئ الفوتوشوب .
- فلاش ٨ .
- البرمجة باستخدام Action script 2.0
- اعداد برامج تعليمية باستخدام الفلاش.
- اعداد العاب باستخدام الفلاش.

دبلومة الجرافيك المستوى الثانى

سعر الدبلومة ٥٠٠ جنية

- برنامج فلاش الأصدار ٩ .
- البرمجة باستخدام Action script 3.0
- برنامج flex 2.0

دبلومة الأوفيس ٢٠٠٧ للمحاسبين

سعر الدبلومة ٥٠٠ جنية

Microsoft Excel 2007 advanced

للمحاسبين شرح كافة الدوال المحاسبية و اعداد قيود اليومية و الحسابات و الميزانية .

Microsoft Access 2007advanced

البرمجة بلغة VBA لعمل برنامج ادارة مخازن و الحصول على **برنامج مراقبة مخازن مجانا**

المراجع

Programming Flex™ 2

ActionScrpt 3 Cookbook

Essential ActionScript 3